

Quick answers to common problems

Puppet 2.7 Cookbook

Build reliable, scalable, secure, high-performance systems to fully utilize the power of cloud computing

John Arundel

目錄

Puppet 2.7 Cookbook 中文版

John Arundel

<hello@bitfieldconsulting.com>

version 1.0, October, 2011

> **Chinese translation version only for learning purposes, do not for commercial purposes. Please buy the original English version.**
>
> 中文翻译版仅用于学习目的，请勿用于商业目的。 支持原作者，请购买英文原版。

中文翻译版

译者： Osmond Liang <linuxbooksAT126DOTcom>

版本： 0.99 （2012/04/30）

中文翻译版使用 AsciiDoc + Github 的方式制作。

- 中文翻译版的源码位于： https://github.com/sinosmond/puppet-27-cookbook-CN
- 您可以用 **GitHub** 的方法参与本书的纠错
- 原书为封闭版权（而非开放版权）的作品
  - 鉴于版权原因，我未在 Github 上以 Github Page 形式发布 HTML 版
  - HTML 翻译版的打包文件发布在 http://down.51cto.com/4139732/
  - 未制作 PDF/epub/mobi 版本，有兴趣的读者可以下载源码自行制作

> 请下载 HTML 翻译版的打包文件，解压缩后在本机阅读
>
> 由于您可以获取翻译源码，鉴于版权原因请不要在公网上直接发布任何 **HTML/PDF/epub/mobi** 版本的中文翻译

译者序

您正阅读的是《Puppet 2.7 Cookbook》的简体中文翻译版。

项目缘起

Puppet 是一款优秀的自动化系统管理部署工具。

去年底曾经在网络上看到过网友 **Sky** 基于《Puppet 2.7 Cookbook》征求意见版（**RAW**）的发布在 BLOG 上的翻译。

- 网友 Sky 的微博： http://weibo.com/u/1938768691
- BLOG 发布： http://www.mysqlops.com/category/puppet

基于 BLOG 的翻译不利于协同翻译、纠错以及追踪原书的版本更新。

今年年初我看到了 蒋鑫 所著的开源书 GotGitHub ，觉得基于 github 写作开放的技术图书或协同翻译图书是个不错的选择。

今年二月无意中从电驴下载了 《Puppet 2.7 Cookbook》 的正式版，遂决意以更利于协同工作的方式重新翻译此书。 并为 **Puppet** 中文社区或其他社区翻译技术图书提供一个案例。

翻译方法

1. 为了协同翻译或协同纠错，应将图书原稿纳入版本控制系统，我选择使用 github 。有关 Github 的使用，请参考 蒋鑫 所著的开源书 GotGitHub 。
2. 显然将 PDF/Word 文档纳入 VCS 的意义不大，为了便于协同工作，应选用一种标记语言书写图书或译作的源码，之后生成 HTML/PDF 文档。 我选择使用 AsciiDoc ，它是一款优秀的基于标记语言的技术文档写作和翻译工具。 为此，我首先将原书的 PDF 文档手工转化成了 AsciiDoc 格式。
3. AsciiDoc 支持 GNU source-highlight 和 Pygments 两种代码加亮系统。 由于当前只有 Pygments 支持 *.pp 代码加亮， 所以我选择使用基于 Pygments 的 puppet-pygments-lexer 对翻译版进行代码加亮处理。
4. 作为一个翻译图书案例，本书所有的翻译工作都是我一个人做的。若要多人协同工作可以采用如下方式：
    - 若要协同翻译，可以开启 Github 中图书翻译项目仓库的 Wiki，进行翻译章节的认领和进度规划等工作。
        - 参考： 使用 Github Wiki
        - 参考： Dive into Python 翻译项目 的 Wiki 组织形式
    - 翻译成员可以在 Github 上克隆核心成员的仓库，在自己的仓库中翻译，之后提交到核心成员的仓库。
        - 参考： Fork + Pull模式
    - 审校者也可用上述方式提交修改，还可以用 缺陷跟踪 方式提交修改意见
5. 使用 asciidoc 或 a2x 生成 HTML/PDF/epub 等版本。

Git and Github Useful Links

- Git
    - git - 简易指南
    - Why Git is Better than X
    - Git Magic
    - git ready
    - Git: Everyday Stuff
    - Everyday GIT With 20 Commands Or So
    - Pro Git
    - 参与 Git 的中文本地化
- Github
    - GotGitHub
    - 如何高效利用 GitHub
    - The GitHub Help

社区链接

- Puppet 中文 Wiki
- Puppet 中文技术社区

社区建议

1. 关于 Puppet 文档的协同翻译建议
    i. 直接使用 Github Wiki 协同翻译（Puppet 文档是 Markdown 格式的，Github Wiki 默认支持 Markdown 格式）
    ii. 使用 Jekyll 创建基于 github 的静态站点生成系统协同翻译 Puppet 文档（参见：使用 Jekyll 维护网站）
2. 关于 Puppet 图书的协同翻译建议
    i. 建议以本书的翻译方式组织 《Pro Puppet》 一书的中文翻译

> 建议 Puppet 中文 Wiki 从 DokuWiki 转向 **Github+Jekyll**。

贡献者

- me
- you
- him
- her

原书版权页

Puppet 2.7 Cookbook

First published: October 2011

Production Reference: 1171011

Published by Packt Publishing Ltd.

Livery Place

32 Lincoln Road

Birmingham, B27 6PA, UK.

ISBN 978-1-84951-538-2

www.packtpub.com

Cover Image by Sujay Gawand (sujay0000@gmail.com)

关于作者

John Arundel is a consultant engineer who helps people build better infrastructure. He uses automation and configuration management to make computer systems cheaper, faster, and more reliable. Formerly a senior enterprise systems engineer in the hosting division of US telco Verizon, he now runs his own company, Bitfield Consulting, and says he has never worked so hard in his life, or for less money.

Over the years John has worked with clients in the advertising and media industry, software, finance, retail, logistics, and even the emergency services, advising on architecture, automation, security, backups, resilience, performance, capacity planning, and regulatory compliance. He has been a member of the Puppet community since its earliest days, and organizes regular local sysadmin meetups and social events.

John holds a B.Sc.(Hons) in Computer Science, with a research interest in kernel resource scheduler design, and is a certified Sun Solaris administrator, LPI (Linux Professional Institute) graduate, and a member of the British Computer Society (MBCS). He is security-cleared to work on computer systems for the UK nuclear industry, which is probably nothing to worry about.

He has also worked as a software developer, both professionally and for the fun of it, contributing to several open source projects, and building a high-performance research chess engine. He blogs regularly at http://bitfieldconsulting.com on Puppet and system administration topics, is usually to be found on Twitter (@bitfield) complaining about things, and often speaks at technical user groups and conferences.

In his negligible spare time, John enjoys repairing Land Rovers, playing Go, and barbecuing. He lives in London and Cornwall.

My thanks go to Luke Kanies and the team at Puppet Labs; also to Ken Barber, Lindsay Holmwood, Gary Larizza, Stephen Nelson-Smith, R.I. Pienaar, Julian Simpson, Jordan Sissel, Cosimo Streppone, James Turnbull, and Dean Wilson, who all provided valuable contributions to the book, whether they know it or not; and for their brave self-sacrifice in the cause of proofreading, Ian Chilton, Kris Buytaert, Stefan Goethals, and Martin Brooks. A special mention goes to the regulars of channel #puppet, who often helped out when things didn't work the way they were supposed to, which was virtually all the time.

前言

IT 运维的一场革命来了。配置管理工具可以在几秒钟内建立服务器并自动化管理你的整个网络。 要充分利用云计算的力量，建立可靠、可扩展、安全、高性能的系统，像 Puppet 这样的工具是必不可少的，

本书为你带来基础知识的基础上，探讨了 Puppet 的强大功能， 向你展示了如何解决各种现实世界问题和应用的详细步骤。 每一个步骤，向你展示了需要键入什么命令，其中还包括每个处方的完整代码样本。

本书从 Puppet 的基础知识入手，为读者介绍了 Puppet 最新最先进的功能， 使读者对 Puppet 有更深入的理解。同时介绍了 Puppet 社区的最佳实践、如何书写优质的配置清单代码、 扩展 Puppet 的部署规模、提高 Puppet 性能、以及如何为 Puppet 添加自定义的提供者和资源等内容。

本书包含了一些来自生产系统的实例，这些实例介绍了如今世界大量 Puppet 安装正在使用的技术， 包括一个分布式的 Puppet 架构和一个使用 Apache 和 Passenger 实现的高性能 Puppetmaster 解决方案。 通过这个实践指南探索世界上最流行的配置管理系统（Puppet）的强大功能。

本书内容

Chapter 1, Puppet Infrastructure introduces some key techniques for managing your Puppet server and manifests, including version control, automated deployment, file serving, pre-signing and autosigning certificates, scaling with Passenger, and a distributed decentralized Puppet architecture using Git.

Chapter 2, Monitoring, Reporting, and Troubleshooting covers ways that Puppet can report information about what it's doing, and the status of your systems. This includes graphical and e-mail reports, log and debug messages, dependency graphing, testing and dry-running your manifests, using tags, run stages, and environments, and a guide to some of Puppet's more common error messages.

Chapter 3, Puppet Language and Style will show you examples of good programming style in Puppet and language constructs that can help you keep your code concise and readable, including conditionals, selectors, case statements, arrays, and regular expressions.

Chapter 4, Writing Better Manifests takes you through structuring your Puppet manifests using node and class inheritance, resource dependencies, and parameterized classes. You'll also see how to get data in and out of Puppet from the environment using CSV files and shell scripts.

Chapter 5, Working with Files and Packages covers powerful techniques for managing config files, including ERB templates, generating files from snippets, and using the Augeas tool. You'll also see how to use Puppet to install packages from APT repositories, and how to set up your own APT and Gem repositories.

Chapter 6, Users and Virtual Resources explains how virtual resources can help you manage different combinations of users and packages on different machines, and shows you how to use Puppet's resource scheduling and auditing features.

Chapter 7, Applications focuses on some specific applications that you may need to manage with Puppet, including complete recipes for Apache and Nginx, MySQL, Drupal, and Rails.

Chapter 8, Servers and Cloud Infrastructure extends the power of Puppet to managing virtual machines, both in the cloud and on your desktop, with recipes for Vagrant and EC2 instances. It also shows you how to set up a Nagios monitoring server, load balancing with HAProxy, firewalls with iptables, network filesystems with NFS, and high-availability services with Heartbeat.

Chapter 9, External Tools and the Puppet Ecosystem looks at the tools that have grown up around Puppet and help you integrate it with the rest of your network, including Puppet Dashboard, Foreman, and MCollective. It also introduces you to some advanced topics including writing your own resource types, providers, and external node classifiers.

阅读前提

要运行这本书中的例子，你需要一台安装了 Ubuntu Linux 10.04 的电脑，安装 Puppet 并与互联网连接。

虽然不是绝对必要的，但我还是建议准备一台咖啡机或其他形式的含咖啡因的饮料机。

适用读者

本书假定你已经安装过 Puppet，或许已经写过一些基本的配置清单或适于发布的模块。 同时需要一些 Linux 系统的管理经验，包括熟悉的命令行、文件系统和文本编辑等。 但没有编程经验的要求。

格式约定

在这本书中，你会发现一些不同的文本样式以区分不同种类的信息。这里是一些样式的例子及其含义。

正文中的代码样式举例；“你需要一台 Puppetmaster 以及 /etc/puppet 目录下的一些已存在的配置清单。”

一个代码块的样式举例：

```
#!/bin/sh

syntax_errors=0
error_msg=$(mktemp /tmp/error_msg.XXXXXX)

if git rev-parse --quiet --verify HEAD > /dev/null
then
against=HEAD
fi
```

一个命令行输入或输出的样式举例：

```
# puppet parser validate/etc/puppet/manifests/site.pp
err: Could not parse for environment production: Syntax error at er
of file at /etc/puppet/manifests/site.pp:3
```

新术语和重要文字 以粗体显示。你在屏幕上看到的文字，例如在菜单或对话框中出现的文字样式举例：“单击 **Next** 按钮进入下一个屏幕”。

> 警告或重要信息的样式。
>
> 技巧和窍门的样式。

读者反馈

Feedback from our readers is always welcome. Let us know what you think about this book?—?what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to feedback@packtpub.com, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on www.packtpub.com or e-mail suggest@packtpub.com.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on www.packtpub.com/authors.

客户支持

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

下载案例代码

You can download the example code files for all Packt books you have purchased from your account at http://www.PacktPub.com. If you purchased this book elsewhere, you can visit http://www.PacktPub.com/support and register to have the files e-mailed directly to you.

勘误表

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books?—?maybe a mistake in the text or the code?—?we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting http://www.packtpub.com/support, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from http://www.packtpub.com/support.

Puppet 基础设施

> Computers in the future may have as few as 1,000 vacuum tubes and weigh only 1.5 tons.
>
> — Popular Mechanics (1949)

在本章中，我们将学习如下内容：

- 使用版本控制
- 使用提交钩子
- 使用 Rake 部署变更
- 配置 Puppet 的文件服务器
- 从 cron 运行 Puppet
- 使用自动签名
- 预签名证书
- 从 Puppet 的 filebucket 检索文件
- 使用 Passenger 扩展 Puppet 的部署规模
- 创建去中心化的分布式 Puppet 架构

本书中的一些处方体现了 Puppet 社区约定的最佳实践。 其他的一些处方是一些技巧和窍门，这些技巧使你使用 Puppet 工作变得更容易， 或向你介绍一些先前还不知道的功能。 其中的一些处方仅仅是轻量级的捷径方法，我不建议你将其作为生产环境的标准操作过程， 但在紧急情况下可能还是有用的。 最后，这里还提供了一些实验性的处方，你可能想尝试一下， 但这些处方仅适用于大型基础设施或其他不寻常的场合。

我希望你通过阅读和学习这里提供的处方，能获得有关 Puppet 如何工作以及你该如何使用 Puppet 的更深入、更广泛的理解， 从而帮助你建立更完善的基础设施。 只有你自己才能决定一个特定的处方对你或你的组织是否适用， 但我希望这里提供的处方能激励你进行实验，找出更多更重要的处方?—?从中获得使用 Puppet 的乐趣！

> 纵观这本书的所有例子，你会看到大多数的命令以 root 用户运行。 如果你更愿意使用一个普通用户账号和 sudo 管理系统， 请使用这种方式替代 root 用户的操作。

因为不同 Linux 发行版（例如：Ubuntu、Red Hat、CentOS）在具体细节上会有不同，比如： 软件包名、配置文件的路径，等等。为了节省篇幅和提高清晰度的原因， 我决定为本书挑选一种发行版（Ubuntu 10.04 Lucid）并坚持使用它。 然而，Puppet 几乎可以在所有 Linux 发行版上运行，所以在你偏爱的操作系统和发行版上应该很少遇到麻烦， 仅需要对处方做适当调整即可。

在这本书写作时，Puppet 2.7 是最新的稳定版本，因此我选择了它作为本书使用的 Puppet 的参考版本。 然而，Puppet 命令的语法每隔一段时间可能就会改变， 所以要注意，旧版本的 Puppet 仍然是非常有用的， 旧版本的 Puppet 可能不支持这本书中所描述的所有功能和语法。

使用版本控制

> Unix was not designed to stop you from doing stupid things, because that would also stop you from doing clever things.
>
> — Doug Gwyn

你曾经遇到过误删除了某些文件而又希望恢复的情形吧？ 本书中提及的最重要的技巧就是将 Puppet 的 配置清单（manifests）纳入像 Git 或 Subversion 这样的 版本控制系统（**Version Control System**，**VCS**）。 直接在 **Puppetmaster** 上编辑配置清单并非明智之举，因为在你还没确信应用这些更改之前可能已经被应用。Puppet 会自动检测配置清单的变化，因此可能会将配置清单的半成品应用到客户端。 这可是个令人讨厌的结果。

取而代之的正确做法是：使用 VCS （我推荐 Git）并从一个版本仓库导出（checkout） Puppetmaster 上所需的 /etc/puppet 目录内容。这样做有如下好处：

- 你不用担心 Puppet 会应用未完成的配置清单
- 你可以撤消更改，并将配置清单恢复到以前的任何版本
- 你可以使用分支（branch）来尝试使用新功能，而不会影响到生产线上使用的主版本（master version）
- 如果多个人需要修改配置清单，他们可以彼此独立的工作，修改他们自己的工作副本，之后合并（merge）他们的改动
- 你可以使用日志来查看何时，何人改动了什么

准备工作

你需要一个运行 Puppetmaster 的主机并且配置清单文件集存放在 /etc/puppet 目录。 如果你还没有准备好这些，可以参考 Puppet 文档（http://docs.puppetlabs.com/）： 如何安装 Puppet（http://docs.puppetlabs.com/guides/installation.html）以及如何建立自己的第一个配置清单 （http://docs.puppetlabs.com/guides/setting_up.html）。

然后将你的配置清单纳入版本控制，可以从 Pupppetmaster 上的 /etc/puppet 目录导入到版本控制系统中, 并使用它作为工作副本。在本例中，我们将使用一个 **GitHub** 账号存储 Puppet 的所有配置。

你需要一个 GitHub 账号（可以免费注册）和并创建一个仓库。跟随 www.github.com 的指示去创建一个吧。

译者注

可以参考 蒋鑫 撰写的 GotGitHub 学习 GitHub 的使用。

你可以使用你的账号从 http://www.PacktPub.com 下载你购买的 Packt 出版的所有书籍的案例代码文件。 如果你从其他地方购买了本书，你可以访问 http://www.PacktPub.com/support ， 注册并直接从 e-mail 获得案例代码文件。

操作步骤

1. 将 Puppetmaster 上的 /etc/puppet 目录纳入一个 Git 仓库，执行如下命令:

```
root@cookbook:/etc/puppet# git init
Initialized empty Git repository in /etc/puppet/.git/
root@cookbook:/etc/puppet# git add manifests/ modules/
root@cookbook:/etc/puppet# git commit -m "initial commit"
[master (root-commit) c7a24cf] initial commit
 59 files changed, 1790 insertions(+), 0 deletions(-)
 create mode 100644 manifests/site.pp
 create mode 100644 manifests/utils.pp
...
```

2. 关联你的 GitHub 仓库并执行推送（push）:

```
# git push -u origin master
Counting objects: 91, done.
Compressing objects: 100% (69/69), done.
Writing objects: 100% (91/91), 21.07 KiB, done.
Total 91 (delta 4), reused 0 (delta 0)
To git@github.com:bitfield/puppet-demo.git
 * [new branch] master -&gt; master
```

分支 master 设置为从 origin 跟踪远程分支 masters。

工作原理

你已经在 GitHub 上创建了一个 “master” 仓库（**repository**）（通常简写为 **repo**）， 它包含了你的 Puppet 配置清单。 你可以在不同的地方导出多个副本，提交变更之前在这些副本上工作。 例如, 如果你有一个系统管理员的团队，他们每个人都可以在他们自己的本地仓库副本上工作。

Puppetmaster 上的 /etc/puppet 目录仅是从属于 GitHub 仓库的另外一个工作副本。 当你决定要在 /etc/puppet 的工作副本上应用 GitHub 仓库上的变更时， 你可以更新这个本地副本，从 GitHub 仓库上获取（pull）最近的变更。

更多用法

既然已经配置了版本控制，你就可以使用如下的工作流程编辑你的 Puppet 配置清单了：

1. 使用自己偏爱的文本编辑器在你自己的工作副本上修改 Puppet 配置清单。

   例如：在我的笔记本电脑上自己的工作副本上添加新的配置清单文件（或做一些编辑工作）：

```
john@laptop:~$ cd puppet-work
john@laptop:~/puppet-work$ mkdir manifests
john@laptop:~/puppet-work$ touch manifests/nodes.pp
john@laptop:~/puppet-work$ git add manifests/nodes.pp
```

2. 提交（commit）变更并推送（push）变更至 GitHub 仓库。

```
john@laptop:~/puppet-work$ git commit -m "adding nodes.pp"
[master 5c7b94c] adding nodes.pp
 0 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 manifests/nodes.pp
john@laptop:~/puppet-work$ git push
Counting objects: 7, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 409 bytes, done.
Total 4 (delta 1), reused 0 (delta 0)
To git@github.com:bitfield/puppet-demo.git
    c7a24cf..b74d452 master -&gt; master
```

3. 使用 git pull 从 Github 仓库更新 Puppetmaster 上的工作副本。

```
root@cookbook:/etc/puppet# git pull
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 4 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From git@github.com:bitfield/puppet-demo.git
    26d668c..5c7b94c master -&gt; origin/master
Updating 26d668c..5c7b94c
Fast-forward
0 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 manifests/nodes.pp
```

你可以使用像 **Rake** 这样的工具实现这一过程的自动化。

参见本书

- 本章的 使用 Rake 部署变更 一节
- 本章的 创建去中心化的分布式 Puppet 架构 一节
- 本章的 使用提交钩子 一节

使用提交钩子

如果我们在提交配置清单之前能够发现其中的语法错误将会是个好消息。 检查 Puppet 配置清单的语法可以使用 puppet parser validate 命令：

```
# puppet parser validate /etc/puppet/manifests/site.pp
err: Could not parse for environment production: Syntax error at er
file at /etc/puppet/manifests/site.pp:3
```

这种语法检查是特别有用的，因为配置清单中的任何一个错误将使运行 Puppet 的节点停止工作， 甚至不会应用配置清单中错误代码的节点也会停止工作。

因此，导入（check in）一个有错误的配置清单，将导致 Puppet 应用这些更新到生产环境使其停止运行， 直到问题被发现，这可能会产生严重的后果。

避免错误发生的最好方法是在你的版本控制仓库中使用 **pre-commit hook** 自动执行语法检查。

操作步骤

如果你使用的是 Git 版本控制系统，可以添加一个脚本 .git/hooks/pre-commit 用于为你提交的所有配置清单文件执行语法检查。 下面的脚本示例来自 Puppet Labs wiki: http://projects.puppetlabs.com/projects/puppet/wiki/ 。

```
#!/bin/sh

syntax_errors=0
error_msg=$(mktemp /tmp/error_msg.XXXXXX)

if git rev-parse --quiet --verify HEAD > /dev/null
then
  against=HEAD
else
    # Initial commit: diff against an empty tree object
    against=4b825dc642cb6eb9a060e54bf8d69288fbee4904
fi

# Get list of new/modified manifest and template files to check (ir
for indexfile in `git diff-index --diff-filter=AM --name-only --cac
  $against | egrep '\.(pp|erb)'`
do
    # Don't check empty files
    if [ `git cat-file -s :0:$indexfile` -gt 0 ]
    then
 case $indexfile in
            *.pp )
                # Check puppet manifest syntax
```

```
                    #git cat-file blob :0:$indexfile | puppet \
                    # --color=false --parseonly --ignoreimport > $erro
                    # Updated for 2.7.x
                    puppet parser validate $indexfile > $error_msg
                    ;;
                *.erb )
                    # Check ERB template syntax
                    # -P : ignore lines which start with "%"
                    git cat-file blob :0:$indexfile | erb -P -x -T - |\
                      ruby -c 2> $error_msg > /dev/null
                    ;;
            esac
   if [ "$?" -ne 0 ]
            then
    echo -n "$indexfile: "
                cat $error_msg
                syntax_errors=`expr $syntax_errors + 1`
            fi
   fi
  done

  rm -f $error_msg

  if [ "$syntax_errors" -ne 0 ]
  then
    echo "Error: $syntax_errors syntax errors found, aborting commit
      exit 1
  fi
```

工作原理

这个提交钩子脚本 .git/hooks/pre-commit 会防止你提交任何有语法错误的配置清单文件：

```
# git commit -m "spot the deliberate mistake" manifests/site.pp
err: Could not parse for environment production: Syntax error at e
file; expected '}' at /etc/puppet/manifests/site.pp:3
manifests/site.pp: Error: 1 syntax errors found, aborting commit.
```

更多用法

你可以在 Puppet Labs wiki: http://projects.puppetlabs.com/projects/1/wiki/Puppet_Version_Control 上找到有关这个脚本的更多详细信息。

你也可以使用类似的 update 钩子防止有语法错误的配置清单推送到 Puppetmaster： 查看同样的 wiki 页面也可以获得使用 update 钩子的详细信息。

参见本书

- 本章的 使用版本控制 一节

使用 Rake 部署变更

每个人的生活都离不开键盘，但我讨厌不必要的敲打。 如果你按照 使用版本控制 一节所描述的工作流程工作， 可以添加一些自动化任务使这个处理过程更加容易。有许多工具可以帮助我们在远程机器上执行命令，包括 Capistrano（https://github.com/capistrano/capistrano）和 Fabric（https://github.com/fabric/fabric），但在本例中，我们将使用 Rake（http://rake.rubyforge.org/）。

准备工作

若你还没有安装 Rake， 运行如下的命令：

```
apt-get install rake
```

执行这个命令，你需要一个 Internet 连接。

操作步骤

1. 在你的 Puppet 配置清单工作副本的顶级目录下创建一个名为 Rakefile 的文件，例如：

   ```
   john@laptop:~/puppet-work$ vi Rakefile
   ```

   文件内容看上去像这样：

   ```
   PUPPETMASTER = 'cookbook'
   SSH = 'ssh -t -A'

   task :deploy do
       sh "git push"
       sh "#{SSH} #{PUPPETMASTER} 'cd /etc/puppet && sudo git pull
   end
   ```

2. 当你在本地副本上应用 Puppet 配置清单的改动时，可以简单地运行命令：

   ```
   $ rake deploy
   ```

3. Rake 会更新远程 Git 仓库并刷新 Puppetmaster 的工作目录副本：

```
$ git push
Counting objects: 4, done.
Delta compression using 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 452 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To ssh:/ /git@cookbook.bitfieldconsulting.com/var/git/cookbook
    561e5a6..a8b8c76 master -&gt; master
ssh -A -l root cookbook 'cd /etc/puppet && git pull'
From ssh://cookbook.bitfieldconsulting.com/var/git/cookbook
    561e5a6..a8b8c76 master -&gt; origin/master
Updating 561e5a6..a8b8c76
Fast-forward
 Rakefile | 6 ++++++
 1 files changed, 6 insertions(+), 0 deletions(-)
 create mode 100644 Rakefile
```

4. 你还可以添加一个 Rake 任务用于在客户端上运行 Puppet：

```
task :apply =&gt; [:deploy] do
    client = ENV['CLIENT']
    sh "#{SSH} #{client} 'sudo puppet agent --test'" do |ok,
    status|
        puts case status.exitstatus
            when 0 then "Client is up to date."
            when 1 then "Puppet couldn't compile the manifest."
            when 2 then "Puppet made changes."
            when 4 then "Puppet found errors."
        end
    end
end
```

5. 当你要在指定客户端上测试你所做的变更时，可以运行如下的命令：

```
rake CLIENT=cookbook apply
```

替换 cookbook 为你客户端的名字，或设置 CLIENT 环境变量，使 Rake 知道你要在哪个客户机上运行 Puppet。

```
info: Caching catalog for cookbook
info: Applying configuration version '1292865016'
info: Creating state file /var/lib/puppet/state/state.yaml
notice: Finished catalog run in 0.03 seconds
```

6. 如果你只想查看 Puppet 将会做些什么，而不是实际应用这些变更， 你可以使用 --noop 标志：

```
task :noop =&gt; [:deploy] do
    client = ENV['CLIENT']
    sh "#{SSH} #{client} 'sudo puppet agent --test --noop'"
end
```

7. 现在你可以运行：

```
$ rake noop
```

   这将会显示一个变更预览。

工作原理

一个 Rakefile 文件由若干任务组成，任务由关键字 task 来标识。 任务定义了一系列的操作步骤，在本例中，使用一系列的 shell 命令推送你的配置清单到主版本仓库， 然后更新 Puppetmaster 上的工作副本。

任务可以互相引用，因为一个任务可能会依赖其它任务。例如，在我们的 Rakefile 中， apply 任务引用了 deploy 任务，每当你运行 rake apply ，Rake 会先确保完成 deploy 任务，然后再执行 apply 任务。

更多用法

你可以扩展这个 Rakefile 实现更多的自动化任务，包括在更新 Puppet 配置清单之前运行语法检查, 甚至可以在引导启动过程中使用 Puppet 来初始化你的新机器。Rake 是一个功能强大的工具，在使用 Puppet 管理大型网络时会为我们提供很大的帮助。

参见本书

- 本章的 使用版本控制 一节
- 本章的 创建去中心化的分布式 Puppet 架构 一节
- 本章的 使用提交钩子 一节

配置 Puppet 的文件服务器

部署配置文件是 Puppet 最常见的用途之一。许多服务都需要一些配置文件， 你可以让 Puppet 使用 file 资源将这些配置文件推送到客户端，如下面的代码所示：

```
file { "/opt/nginx/conf.d/app_production.conf":
    source => "puppet:///modules/app/app_production.conf",
}
```

source 参数是这样约定的： puppet:/// 之后的第一部分假定是一个 挂装点（**mount point**） 名称，其余部份被视为一个文件路径，如下所示：

```
puppet:///<mount point>/<path>
```

通常 <mount point> 的值是一个模块名称，如上例所示。在这个例子中， Puppet 将在如下的位置查找文件：

```
manifests/modules/app/files/app_production.conf
```

modules 是 Puppet 予以特别对待的一个挂装点：它期望接下来的路径组成是一个模块名， 并在模块的 files 目录下针对路径的其余部分寻找文件。

然而 Puppet 也允许你创建自定义的挂装点，你可以为自定义的挂装点设置个别的访问控制， 并将其映射到 Puppetmaster 的不同文件系统位置。 在本节中，我们将展示如何创建和配置这些自定义的挂载点。

操作步骤

1. 在 PuppetMaster 的 fileserver.conf 中添加新的一节，将挂装点的名称用方括号括起， path 的值就是 Puppet 将会寻找数据的目录路径，如下所示：

    ```
    [san]
        path /mnt/san/mydata/puppet
    ```

2. 在你的配置清单里，使用 source 指定你的挂载点名称，如下所示：

    ```
    source => "puppet:///san/admin/users.htpasswd",
    ```

    Puppet 会将其转换为如下的路径：

```
/mnt/san/mydata/puppet/admin/users.htpasswd
```

像这样创建一个自定义挂载点的主要原因就是提高安全性。 例如，你有个秘密的口令文件只需部署到 web 服务器，而其它机器则不需要。 如果有人能够在任何机器上运行 Puppet，并且有合法的证书访问 Puppetmaster， 那么没有人能阻止他像这样执行下面的配置清单：

```
file { "/home/cracker/goodstuff/passwords.txt":
    source =&gt; "puppet:///web/passwords.txt",
}
```

他们可以轻而易举地获取秘密数据。事实上，可以导出 Puppet 仓库的任何人以及 在 Puppetmaster 上有账户的任何人都可以访问此文件。 为了避免这种情况发生的方法之一就是将秘密数据放在自定义挂载点并启用访问控制。

3. 在 fileserver.conf 中添加 allow 和 deny 参数来定义你的挂载点，如下所示：

```
[secret]
    /data/secret
    allow web.example.com
    deny *
```

工作原理

在本例中, 仅允许 web.example.com 访问此文件。 默认的策略是拒绝所有的访问， 因此 deny * 这行是可选的，但它确实是个好的习惯，因为看上去更清晰。 之后 web 服务器就可以使用 file 资源了，如下所示：

```
file { "/etc/passwords.txt":
    source => "puppet:///secret/passwords.txt",
}
```

如果此配置清单是在 web.example.com 上执行，将会正常工作； 若在其它客户端上执行则执行失败。

更多用法

你也可以将指定的 IP 地址替换为主机名，也可使用 无类型域间路由**(CIDR)** 或使用通配符（wildcard）来表示一组地址，如下所示：

```
allow 10.0.55.0/24
allow 192.168.0.*
```

参见本书

- 第 3 章的 使用模块 一节
- 第 6 章的 使用文件资源递归地分发整个目录树 一节
- 第 6 章的 为文件资源指定多个源 一节

从 cron 运行 Puppet

你的 Puppet 工作在休眠状态吗？ 默认情况下，当你在客户端上运行 Puppet agent 时， 它会以守护进程（后台进程）的方式执行，每隔30分钟唤醒一次并检查配置清单是否有更新并应用这些变更 （也可以在 puppet.conf 中将 splay 选项的值设为 true 来指定一个随机的时间间隔）。 如果想要更灵活的控制 Puppet 的运行，你可以安排 cron 任务来替代守护进程的触发执行方式。

例如，如果你有很多的 Puppet 客户端，可能需要刻意地错开 Puppet 的运行时间以减轻 Puppetmaster 的负载压力。一个简单的方法是：将客户端主机名的哈希（散列）值作为 cron 任务的分钟或小时的时间参数。

操作步骤

使用 Puppet 的 inline_template 函数，它允许你执行 Ruby 代码：

```
cron { "run-puppet":
    command => "/usr/sbin/puppet agent --test",
    minute  => inline_template("<%= hostname.hash.abs % 60 %>"),
}
service { "puppet":
    ensure => stopped,
    enable => false,
}
```

工作原理

因为每个主机名会生成一个唯一的哈希值，所以每个客户端将会在一小时的不同分钟数运行 Puppet。 这种散列技术对于使 cron 任务随机运行是相当有用的，因为分散了运行时间，改善了互相干扰， 从而减轻了 Puppetmaster 的负载压力。

更多用法

你可能会发现，随着时间的推移以守护进程方式运行的 Puppet 会占用更多的内存， 或者偶尔 Puppet 与 Puppetmaster 的通信处于停滞状态（stuck state）。 以 cron 方式运行 Puppet 会解决这些问题。

参考 Puppet Labs 的 Wiki 页面 http://projects.puppetlabs.com/projects/puppet/wiki/Cron_Patterns 获得更多信息。

触发 Puppet 的运行还有其他方式，包括 **MCollective** 工具， 我们将在这本书的后续章节中做详细介绍。

参见本书

- 第 6 章的 有效地分发 cron 任务 一节
- 第 3 章的 使用嵌入式 Ruby 代码 一节
- 第 9 章的 使用 MCollective 一节

使用自动签名

在密码学中，跟生活一样，在你签名时必须小心慎重。一般地，当你要为 Puppetmaster 介绍一个新的客户端加入时，需要先在客户端上生成一个证书请求（certificate request），然后到 Puppetmaster 上签署这个证书请求。然而，你可以使用 自动签名（**autosigning**） 跳过这一步骤。

操作步骤

在 Puppetmaster 上创建文件 /etc/puppet/autosign.conf ，包含如下内容：

```
*.example.com
```

工作原理

Puppet 会检查所有的证书请求是否匹配 autosign.conf 中任何一行。任何能匹配客户端主机名 *.example.com 的证书请求，Puppetmaster 都会为其自动签名。

> 这里存在一个潜在的安全问题，因为这相当于 Puppetmaster 信任了任何可以连接到它的客户端（只要主机名匹配）。基于这种原因，不推荐你使用自动签名。如果你确信要使用自动签名，请确保 Puppetmaster 被防火墙保护，且只允许信任的客户端或者 IP 地址段连接 Puppetmaster。更安全的做法是使用 预签名（**pre-signing**）。

参见本书

- 本章的 预签名证书 一节

预签名证书

因为存在安全隐患，如果你能辅助签名过程，最好避免使用自动签名。 在一般情况下，如果你想自动加入大量的客户端，最好在 Puppetmaster 上预先生成证书，然后将其作为构建过程的一部分推送给客户端。 你可以使用 puppet cert --generate <hostname> 命令生成预签名证书（pre-signed certificate）。

操作步骤

1. 使用如下命令为 client1.example.com 生成预签名证书：

   ```
   puppet cert --generate client1.example.com
   ```

   Puppet 现在将为客户端 client1.example.com 生成并签署客户端证书。

2. 传输三个所需的文件到新的客户端：包括客户端私钥、客户端证书和 CA 证书。 这三个文件位于：

   ```
   /etc/puppet/ssl/private_keys/client1.example.com.pem
   /etc/puppet/ssl/certs/client1.example.com.pem
   /etc/puppet/ssl/certs/ca.pem
   ```

   复制上述三个文件到客户端相应的目录下，Puppet 会自动进行身份验证从而省略证书请求这一步骤。 值得注意的是 Puppet 的 SSL 证书的位置依赖于 puppet.conf 中的 ssldir 设置。

参见本书

- 本章的 使用自动签名 一节

从 Puppet 的 filebucket 检索文件

> A Freudian slip is when you say one thing, but mean your mother.
>
> — Anon

我们每个人都会犯错误，这就是为什么通常铅笔上会配有橡皮擦的原因。 每当 Puppet 客户端在改变一个文件时，就会将改变前的版本做个备份。 如果在 Puppet 客户端上对一个已经存在的文件做修改，不管多小的改变， 我们都可以看到这一过程：

```
# puppet agent --test
info: Caching catalog for cookbook
info: Applying configuration version '1293459139'
--- /etc/sudoers 2010-12-27 07:12:20.421896753 -0700
+++ /tmp/puppet-file20101227-1927-13hjvy6-0 2010-12-27 07:13:21.645
-0700
@@ -12,7 +12,7 @@

 # User alias specification
-User_Alias SYSOPS = john
+User_Alias SYSOPS = john,bob

info: FileBucket adding /etc/sudoers as {md5}
c07d0aa2d43d58ea7b5c5307f532a0b1
info: /Stage[main]/Admin::Sudoers/File[/etc/sudoers]: Filebucketed
sudoers to puppet with sum c07d0aa2d43d58ea7b5c5307f532a0b1

notice: /Stage[main]/Admin::Sudoers/File[/etc/sudoers]/content: cor
changed '{md5}c07d0aa2d43d58ea7b5c5307f532a0b1' to '{md5}0d218c16bc
312c885884fa947d'

notice: Finished catalog run in 0.45 seconds
```

我们感兴趣的是下面这行：

```
info: /Stage[main]/Admin::Sudoers/File[/etc/sudoers]: Filebucketed
sudoers to puppet with sum c07d0aa2d43d58ea7b5c5307f532a0b1
```

Puppet 会根据文件内容创建一个 MD5 哈希，并使用它来创建一个 **filebucket** 路径， **filebucket** 的值是基于哈希的前几个字符的。 filebucket 用来保存 Puppet 替换下来的任何文件副本，它存放的默认位置是 /var/lib/puppet/clientbucket：

```
# ls /var/lib/puppet/clientbucket/c/0/7/d/0/a/a/2/
c07d0aa2d43d58ea7b5c5307f532a0b1
contents  paths
```

正如你看到的，ls 命令列出了文件名。 你在 bucket 的存放位置会看到两个文件：contents 和 paths 。 contens 文件的内容即为原始文件，paths 文件的内容即为原始文件的路径。

如果你知道文件内容的哈希值（像你看到的上面的例子），可以很容易地找到该文件； 如果你不知道，那么通过对整个 filebucket 创建一个索引文件的表将会非常有用。

操作步骤

1. 使用如下命令创建索引文件：

```
# find /var/lib/puppet/clientbucket -name paths -execdir cat {}
   -execdir pwd \; -execdir date -r {} +"%F %T" \; -exec echo \
   &gt; bucket.txt
```

2. 在索引文件中查找你要寻找的文件：

```
# cat bucket.txt
/etc/sudoers
/var/lib/puppet/clientbucket/c/0/7/d/0/a/a/2/
c07d0aa2d43d58ea7b5c5307f532a0b1
2010-12-27 07:13:21

/etc/sudoers
/var/lib/puppet/clientbucket/1/0/9/0/e/2/8/a/1090e28a70ebaae872
c78894f49eb
2010-12-27 07:12:20
```

3. 一旦你要恢复一个已知 bucket 路径的文件，只要复制该文件到原始文件名即可：

```
# cp /var/lib/puppet/clientbucket/1/0/9/0/e/2/8/a/1090e28a70eba
72c2ec78894f49eb/contents /etc/sudoers
```

工作原理

上面的 find 命令会创建一份完整的 filebucket 文件列表清单， 显示原始文件的名称，bucket 的路径，以及修改日期 （在上例中你学习到了如何恢复文件到以前版本）， 一旦你知道 bucket 的路径，那么就可以复制文件到正确的位置。

更多用法

你可以让 Puppet 在原始目录下创建备份文件，而不是在 filebucket。 为了做到这一点，只需要在配置清单中指定 backup 参数的值：

```
file { "/etc/sudoers":
    mode => "440",
    source => "puppet:///modules/admin/sudoers",
    backup => ".bak",
}
```

现在，如果 Puppet 替换了旧文件，就会在原始路径下面创建一个扩展名为 .bak 的备份文件。 若希望 Puppet 对所有的文件执行这样的默认备份策略，可以在配置清单中使用如下代码：

```
File {
    backup => ".bak",
}
```

要完全禁用备份，使用下面的代码：

```
    backup => false,
```

使用 **Passenger** 扩展 **Puppet** 的部署规模

如果你的 Puppet 基础设施开始出现依依呀呀的裂缝，罪魁祸首很可能出现在 Puppetmaster 的 Web 服务器上。 Puppet 携带了一个名为 **Webrick** 的简单 Web 服务器来处理客户端与 Puppetmaster 的连接。Webrick 确实不适合在生产环境下运行 Puppet； 超过几个客户请求处理之后就会使 Puppetmaster 一蹶不振，严重影响其性能。

有时会建议使用 **Mongrel** 替换 Webrick，因为 **Mongrel** 比 Webrick 有少许的性能提升，但不太明显。为了扩展 Puppetmaster 能服务于数百台服务器， 首选的方法是切换到高性能的 Web 服务器，如使用包含 **Passenger**（mod_rails） 扩展模块的 Apache。

Puppet 在 Passenger 下运行需要些必要的配置，你需要安装 Apache 和 Passenger，并添加一个合适的虚拟主机。下面的示例是基于 Ubuntu 10.4 的。 你可以在 Puppet Labs 的站点 http://projects.puppetlabs.com/projects/1/wiki/Using_Passenger 上找到 Red Hat Linux、CentOS以及其它发行版本的对应操作指示。

准备工作

为了方便配置，需要有你要运行的 Puppet 的源代码包（**tarball**）， 因为它提供了用于配置 Passenger 的一些模板文件和配置片段。 例如，假如你要运行 Puppet 2.7.1，就要下载这个文件： http://puppetlabs.com/downloads/puppet/puppet-2.7.1.tar.gz 。

若你使用不同的版本，请到 http://puppetlabs.com 下载合适的版本。 下载之后使用如下命令解开源码包：

```
tar xzf puppet-2.7.1.tar.gz
```

操作步骤

1. 安装 Apache 和 Passenger, 及其所依赖的软件包：

   ```
   # apt-get install apache2 libapache2-mod-passenger rails
   librack-ruby libmysql-ruby
   # gem install rack
   ```

2. 为 Passenger 查找 Puppet 的配置创建必要的目录：

   ```
   # mkdir -p /etc/puppet/rack
   # mkdir -p /etc/puppet/rack/public
   ```

这两个目录的属主为 root 且权限为 0755。

```
# chown -R root:root /etc/puppet/rack
# chmod -R 0755 /etc/puppet/rack
```

3. 创建文件 config.ru ，此文件告知 Passenger 如何启动 Puppet 应用程序。 你可以使用 Puppet 发布中提供的示例文件：

```
# cp /tmp/puppet-2.7.1/ext/rack/files/config.ru /etc/puppet/rac
# chown puppet /etc/puppet/rack/config.ru
```

对于 Puppet 2.7.1, 应该包含如下内容：

```
# a config.ru, for use with every rack-compatible webserver.
# SSL needs to be handled outside this, though.

# if puppet is not in your RUBYLIB:
# $:.unshift('/opt/puppet/lib')

$0 = "master"

# if you want debugging:
# ARGV &lt;&lt; "--debug"

ARGV &lt;&lt; "--rack"
require 'puppet/application/master'
# we're usually running inside a Rack::Builder.new {} block,
# therefore we need to call run *here*.
run Puppet::Application[:master].run
```

4. 你现在需要在 Apache 上创建一个虚拟主机，使其监听正确的端口并向 Puppet 应用程序发送请求。同样地，你可以使用 Puppet 发布中提供的示例文件：

```
# cp /tmp/puppet-2.7.1/ext/rack/files/apache2.conf \
  /etc/apache2/sites-available/puppetmasterd
# a2ensite puppetmasterd
```

文件的内容如下所示：

```
# you probably want to tune these settings
PassengerHighPerformance on
PassengerMaxPoolSize 12
PassengerPoolIdleTime 1500
# PassengerMaxRequests 1000
PassengerStatThrottleRate 120
RackAutoDetect Off
RailsAutoDetect Off

Listen 8140

&lt;VirtualHost *:8140&gt;
  SSLEngine on
  SSLProtocol -ALL +SSLv3 +TLSv1
  SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:-LOW:-SSLv2:-EX

  SSLCertificateFile /etc/puppet/ssl/certs/cookbook.
  bitfieldconsulting.com.pem
  SSLCertificateKeyFile /etc/puppet/ssl/private_keys/cookbook.
  bitfieldconsulting.com.pem
  SSLCertificateChainFile /etc/puppet/ssl/ca/ca_crt.pem
  SSLCACertificateFile /etc/puppet/ssl/ca/ca_crt.pem
  # If Apache complains about invalid signatures on the CRL, yo
  # can try disabling
  # CRL checking by commenting the next line, but this is not
  # recommended.
  SSLCARevocationFile /etc/puppet/ssl/ca/ca_crl.pem
  SSLVerifyClient optional
  SSLVerifyDepth 1
  SSLOptions +StdEnvVars

  DocumentRoot /etc/puppet/rack/public/
  RackBaseURI /
  &lt;Directory /etc/puppet/rack/&gt;
    Options None
    AllowOverride None
    Order allow,deny
    allow from all
  &lt;/Directory&gt;
&lt;/VirtualHost&gt;
```

5. 编辑这个文件，将 SSLCertificateFile 和 SSLCertificateKeyFile 的值修改为你自己的证书（最简单地生成证书的方法是你已经运行 Puppet 至少一次）。

6. 你还要在 Apache 中启用 Passenger 和 mod_ssl 模块：

```
# a2enmod passenger ssl
```

7. 添加如下的行到你的 /etc/puppet/puppet.conf 文件：

```
ssl_client_header = SSL_CLIENT_S_DN
ssl_client_verify_header = SSL_CLIENT_VERIFY
```

8. 停止正在运行的 Puppetmaster。
9. 启动 Apache：

```
# /etc/init.d/apache2 restart
```

10. 如果一切工作正常，你可以像往常一样运行 Puppet：

```
# puppet agent --test
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1294145142'
notice: Finished catalog run in 0.25 seconds
```

工作原理

Puppet 内置的 web 服务器处理速度相当慢，每次只能处理一个请求，使用 Apache 替换它之后，现在你就可以使用这个高性能多线程的 web 服务器了。 在这种情况下，Puppet 是一个使用 Rack 框架（Rack framework） 的嵌入式应用程序， 这大大提高了运行效率。你应该会发现，使用 "Apache + Passenger" 的配置能处理更多的客户端和更频繁的 Puppet 请求，并且改善了服务器内存的占用， 使用的内存比标准的 Puppetmaster 守护进程占用的内存更少。

更多用法

下面是一个 Puppet 配置清单的示例，用于为你实现上面的处理步骤（基于 Ubuntu 系统）：

```
class puppet::passenger {
    package { [ "apache2-mpm-worker",
                "libapache2-mod-passenger",
                "librack-ruby",
                "libmysql-ruby" ]:
        ensure => installed,
    }

    service { "apache2":
        enable => true,
        ensure => running,
        require => Package["apache2-mpm-worker"],
    }

    package { "rack":
        provider => gem,
        ensure => installed,
    }

    file { [ "/etc/puppet/rack",
             "/etc/puppet/rack/public" ]:
        ensure => directory,
        mode => "755",
    }

    file { "/etc/puppet/rack/config.ru":
        source => "puppet:///modules/puppet/config.ru",
        owner => "puppet",
    }

    file { "/etc/apache2/sites-available/puppetmasterd":
        source => "puppet:///modules/puppet/puppetmasterd.conf",
    }

    file { "/etc/apache2/sites-enabled/puppetmasterd":
        ensure => symlink,
        target => "/etc/apache2/sites-available/puppetmasterd",
    }

    exec { "/usr/sbin/a2enmod ssl":
        creates => "/etc/apache2/mods-enabled/ssl.load",
    }
}
```

一旦你以 Passenger 方式运行，就可以使用如下命令重新启动 Puppetmaster 应用程序：

```
# service apache2 restart
```

为了监视 Passenger 正在运行，可以检查名为 ApplicationPoolServerExecutable 的进程。

你也可以用配置常规 web 应用的方法为 Passenger 实例配置负载均衡。

更多详细信息，或者如果你遇到问题，可以参考 **Puppet-on-Passenger** 文档：http://projects.puppetlabs.com/projects/1/wiki/Using_Passenger 。

参见本书

- 本章的 创建去中心化的分布式 Puppet 架构 一节

创建去中心化的分布式 **Puppet** 架构

> I have the world's largest collection of seashells. I keep it scattered around the beaches of the world… perhaps you've seen it.
>
> — Steven Wright

某些系统（尤其像 Mafia）在去中心化的分布式架构环境下运行良好。 使用 Puppet 的一个最常见的方法是运行一个 Puppetmaster 服务器， Puppet 客户端连接这个服务器并从它获取配置清单。 无论如何，你仍旧可以使用下面的方法直接针对配置清单运行 puppet apply 命令。（通常会使用 -v 参数开启冗余输出模式， 这样就可以看到详细的执行过程）：

```
# puppet apply -v manifest.pp
info: Applying configuration version '1294313350'
```

你甚至可以直接在命令行上运行配置清单的代码片断：

```
# puppet apply -e "file { '/tmp/test': ensure => present }"
notice: /Stage[main]//File[/tmp/test]/ensure: created
```

换言之，如果能安排合适的配置清单并分发到客户端，你就可以在 Puppet 客户端上直接执行它，而不必使用中心化的 Puppetmaster 服务器。 这将消除单台 Puppetmaster 服务器的性能瓶颈，也消除了单点故障。 与此同时，也避免了新增客户端时 SSL 证书签署以及证书交换的步骤。

将配置清单文件推送到客户端有多种实现方法， 显然 Git （或者其它版本控制系统，如 Mercurial 或 Subversion）能为你做绝大部分的工作。 你可以在本地编辑 Puppet 配置清单的本地副本，提交到 Git 并将更新推送到中心仓库， 然后从 Git 中心仓库自动将配置清单分发到客户机。

准备工作

如果你的 Puppet 配置清单还没有纳入 Git，请参考本章 使用版本控制 一节的操作步骤。

操作步骤

1. 在客户端使用如下命令对 Puppet 仓库克隆一个裸仓库（裸仓库没有工作区，只包含 .git 目录中的所有内容）：

   ```
   # git clone --bare ssh://git@repo.example.com/var/git/puppet
   ```

2. 使用如下命令检出裸仓库内容到 /etc/puppet/ 目录：

```
# git archive --format=tar HEAD | (cd /etc/puppet && tar xf -)
```

3. 使用 Puppet 命令执行 site.pp 文件

```
# puppet apply -v /etc/puppet/manifests/site.pp
info: Applying configuration version '1294313353'
```

一旦完成上面的工作，下一步就是配置仓库的自动推送将变动推送到客户端。使用 Git 的 remote 命令可以实现此功能，即配置本地仓库的远程仓库别名。例如：

```
# git remote add web ssh://git@web1.example.com/etc/puppet
```

如果你有多个客户端， 可以为同一个远程别名添加多个 URL：

```
# git remote set-url --add webs ssh://git@web2.example.com/etc/
# git remote set-url --add webs ssh://git@web3.example.com/etc/
...
```

或者像这样，简单编辑 Git 配置文件 (.git/config) ：

```
[remote "web"]
    url = ssh://git@web1.example.com/etc/puppet
    url = ssh://git@web2.example.com/etc/puppet
    url = ssh://git@web3.example.com/etc/puppet
    ...
```

4. 现在你可以使用如下命令从 Git 中心仓库推送更新到任意一台（或一组）客户端：

```
# git push web
```

5. 最后一步，每当客户端接收到从 Git 中心仓库的推送就应该更新它自己的 /etc/puppet 目录。你可以使用 Git 的 postreceive 钩子来实现此功能。 在你的仓库中，创建一个名为 hooks/post-receive 的文件并为其设置可执行权限（0755）， 文件内容为：

```
#!/bin/sh
git archive --format=tar HEAD | (cd /etc/puppet && tar xf -)
```

工作原理

与连接 Puppetmaster 获取已经编译好的客户端配置清单不同，使用这种方式每个客户端都要从本地副本编译自己的配置清单。每当 Git 中心服务器推送一次更新（或者从 Git 仓库 checkout 检出）就会重新编译一次。这大大高了网络带宽的利用率，因为客户端不必每次运行时都连接 Puppetmaster。同时也消除了单点故障，因为客户端可以从任何分布式的 Git 仓库（这些仓库的内容是由 Git 中心仓库自动推送的）获得更新。

使用基于 Git 的去中心化的分布式 Puppet 架构为你提供了一个非常灵活的处理方式。你可以使用 SSH 密钥来配置访问控制，按需要允许每一个客户端或一组客户端的访问。例如：用于数据库服务器的配置清单仅允许数据库组的机器访问获取。

当然还需要一些额外的配置工作，但对于大多数小型组织是没有必要的，这种去中心化的 Puppet 部署方式提供了额外的灵活性，同时也适用于更为苛刻的权限控制环境。

更多用法

如果你希望每次中心仓库推送更新之后 Puppet 立即执行并应用更新，可以编辑 post-receive 脚本来完成，或者执行你需要的其它的动作。另外，你也可以手动运行 Puppet，还可以用 从 cron 运行 Puppet 一节描述的方法使用 cron 调度运行，但要记得是运行 puppet apply ，而不是运行 puppet agent 。

使用基于 Git 的架构也存在一些缺点：不能使用 Puppet 提供的高级特性，例如外部节点分类器（external node classifier）以及存储配置（stored configuration）等。不管怎样，当你需要将 Puppet 的部署扩展到大量节点时，这是一种最简单的实现方式。

你可以在 Stephen Nelson-Smith 的这篇文章里找到更多更详细的关于这种架构的讨论，文章地址是：http://bitfieldconsulting.com/scaling-puppet-with-distributedversion-control 。

参见本书

- 本章的 使用 Passenger 扩展 Puppet 的部署规模 一节
- 本章的 使用版本控制 一节

监控、报告和排错

> Found problem more than one. However, this does not mean that relevant part is thing by mistake. Could be fertilized by special purpose in other application program.
>
> — Error message

在本章中，我们将学习如下内容：

- 生成报告
- 通过 Email 发送包含特定标签的日志信息
- 创建图形化报告
- 自动生成 HTML 文档
- 绘制依赖关系图
- 测试你的 Puppet 配置清单
- 执行模拟运行
- 检测编译错误
- 理解 Puppet 的错误信息
- 显示命令的输出结果
- 输出调试信息
- 检查配置设置
- 使用标签
- 使用运行阶段
- 使用不同的环境

大家都有这样的经验：当看到一些新技术时会莫名的兴奋，急着赶回家进行实验。当然，一旦你开始尝试使用它，马上就会遇到问题。 这是怎么回事呢？它为什么不工作呢？我怎么能看到引擎盖下发生了什么呢？ 本章将帮助你回答这些问题，并为你介绍解决常见的 Puppet 问题的工具。 我们还会看到：如何对你的 Puppet 基础设施生成有用的报告， 以及 Puppet 如何帮你对整个网络进行监视和故障排查。

生成报告

> What the world really needs is more love and less paperwork.
>
> — Pearl Bailey

大型的基础设施架构往往是真正的第一受害者。如果你正管理着许多机器， 那么使用 Puppet 的报告设施，可以为你提供一些关于正在发生什么的有价值的信息。

操作步骤

要启用报告，只要将如下的行加入客户端的 puppet.conf 文件：

```
report = true
```

工作原理

启用报告之后，Puppet 将在 Puppetmaster 上生成报告文件，包含如下的数据：

- 从 Puppetmaster 获取配置所需要的时间
- Puppet 的总体运行时间
- 运行期间的日志消息输出
- 客户端配置清单中的所有资源列表
- Puppet 是否改变了每个资源
- 一个资源是否与配置清单不同步

默认情况下，这些报告存储在 /var/lib/puppet/reports 目录下，你也可以使用 reportdir 参数指定到不同的位置。你可以创建自己的脚本处理这些报告 （这些报告都是标准的 **YAML** 格式），或者使用工具，例如 **Puppet Dashboard** 来获得您的网络图形概览。

更多用法

在下文中将解释从 Puppet 报告中收集信息的几个实用技巧。

从命令行启用报告

如果你只是想要一个报告，或者你不想让所有客户端都发送报告， 你可以切换到命令行，手动执行带有 --report 参数的命令：

```
# puppet agent --test --report
```

你还可以使用如下带有 --summarize 参数的命令看到关于 Puppet 的运行统计概要信息：

```
# puppet agent --test --summarize
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1306169315'
notice: Finished catalog run in 0.58 seconds
Changes:
Events:
Resources:
             Total: 7
Time:
  Config retrieval: 3.65
        Filebucket: 0.00
          Schedule: 0.00
```

将 **Puppet** 的信息记入系统日志

Puppet 也可以将日志信息发送到 Puppetmaster 的 系统日志（**syslog**），因此你可以使用标准的 syslog 工具分析这些日志消息。为了实现这一点，你可以在 Puppetmaster 的配置文件 puppet.conf 中添加如下所示的选项：

```
[master]
reports = store,log
```

默认的报告类型是 store （将报告输出到 /var/lib/puppet/reports），log 选项是告诉 Puppet 同时将消息发送到 syslog。

参见本书

- 本章的 创建图形化报告 一节
- 本章的 输出调试信息 一节
- 第 9 章的 使用 Puppet Dashboard 一节

通过 Email 发送包含特定标签的日志信息

像大多数系统管理员一样，如果没有收到足够的邮件，你会寻找一种方法生成邮件。 另外一种 Puppet 报告形式被称为 tagmail。这会根据你设定的 e-mail 地址将日志信息发送到你指定的邮箱。

操作步骤

1. 在 puppet.conf 文件中为 reports 添加以逗号分割的 tagmail 选项：

```
[master]
reports = store,tagmail
```

2. 在 /etc/puppet/tagmail.conf 文件中，添加一些 标签（**tags**）并指定相应的 e-mail 地址。例如：下面的配置行将所有的日志消息发送到 john@bitfieldconsulting.com ：

```
all: john@example.com
```

3. 一旦 Puppet 运行，你就会收到一份类似如下内容的 e-mail：

```
From: report@cookbook.bitfieldconsulting.com
Subject: Puppet Report for cookbook.bitfieldconsulting.com
To: john@example.com

Mon Jan 17 08:42:30 -0700 2011 //cookbook.bitfieldconsulting.co
Puppet (info): Caching catalog for cookbook.bitfieldconsulting.
Mon Jan 17 08:42:30 -0700 2011 //cookbook.bitfieldconsulting.co
Puppet (info): Applying configuration version '1295278949'
```

工作原理

Puppet 在 tagmail.conf 配置文件中查找每一行配置， 匹配标签（tag）并将消息发送到指定的邮箱。名为 all 的特殊标签会匹配所有的消息。 名为 err 的标签会匹配所有的错误消息：

```
err: john@example.com
```

你可以在 tagmail.conf 文件中定义多个规则， Puppet 会为所有匹配的规则发送邮件。 在下面的例子中，错误消息发送到一个邮件地址， 而与 web 服务器相关的消息发送到另一个邮件地址：

```
err: puppetmaster@example.com
webserver: webteam@example.com
```

更多用法

tagmail 报告是一个非常强大的特性， 你可能需要在实践中才能获得相关的体会。下面列出了一些有用的技巧。

什么是标签？

在 使用标签 一节中将会对标签做更详细的解释，但为了能使用标签的报告特性，现在只要知道 **tag** 可以为节点（node）或类（class）命名就足够了。例如：webserver 标签匹配所有执行 webserver 类的客户端。

你也可以像下面这样使用 tag 函数直接添加一个标签：

```
class exim {
    tag("email")
    service { "exim4":
        ensure => running,
        enable => true,
    }
}
```

指定多个标签或排除指定的标签

你可以在 tagmail.conf 文件中以逗号间隔方式指定一个标签列表， 也可以在某个标签之前使用叹号（!）排除这个标签。

```
all, !webserver: puppetmaster@example.com
```

发送报告到多个 **e-mail** 地址

你可以将消息同时发送到多个地址，用如下的方式使用逗号分割多个 e-mail 地址：

```
err: puppetmaster@example.com, sysadmin@example.com
```

参见本书

- 本章的 生成报告 一节
- 本章的 创建图形化报告 一节
- 本章的 使用标签 一节

创建图形化报告

让我们面对现实，老板们总是喜欢看漂亮的图片。 Puppet 使用 **RRD**（**Round-Robin Database**） 图形库生成适应生产的报告数据，生成图形化的技术指标，如客户端的运行时间等。

准备工作

你需要在系统上安装 RRD 工具以及 Ruby 所需的相关链接库。 对于 Ubuntu 系统需要执行如下命令：

```
# apt-get install rrdtool librrd-ruby
```

操作步骤

添加 rrdgraph 报告类型到你的 puppet.conf 文件：

```
reports = store,rrdgraph
```

工作原理

每次运行，Puppet 会将数据记录到客户端的 RRD 目录 （默认为 /var/lib/puppet/rrd/<clientname>）。 它会为事件（events）、资源（resources）以及获取时间（retrieval time） 创建 PNG 格式的图片，当然若你希望使用第三方的 RRD 工具处理，可以使用以 .rrd 结尾的原始数据文件。

更多用法

要获得关于图形化报告的更详细的信息，你可以使用 Puppet Dashboard。

参见本书

- 第 9 章的 使用 Puppet Dashboard 一节

自动生成 HTML 文档

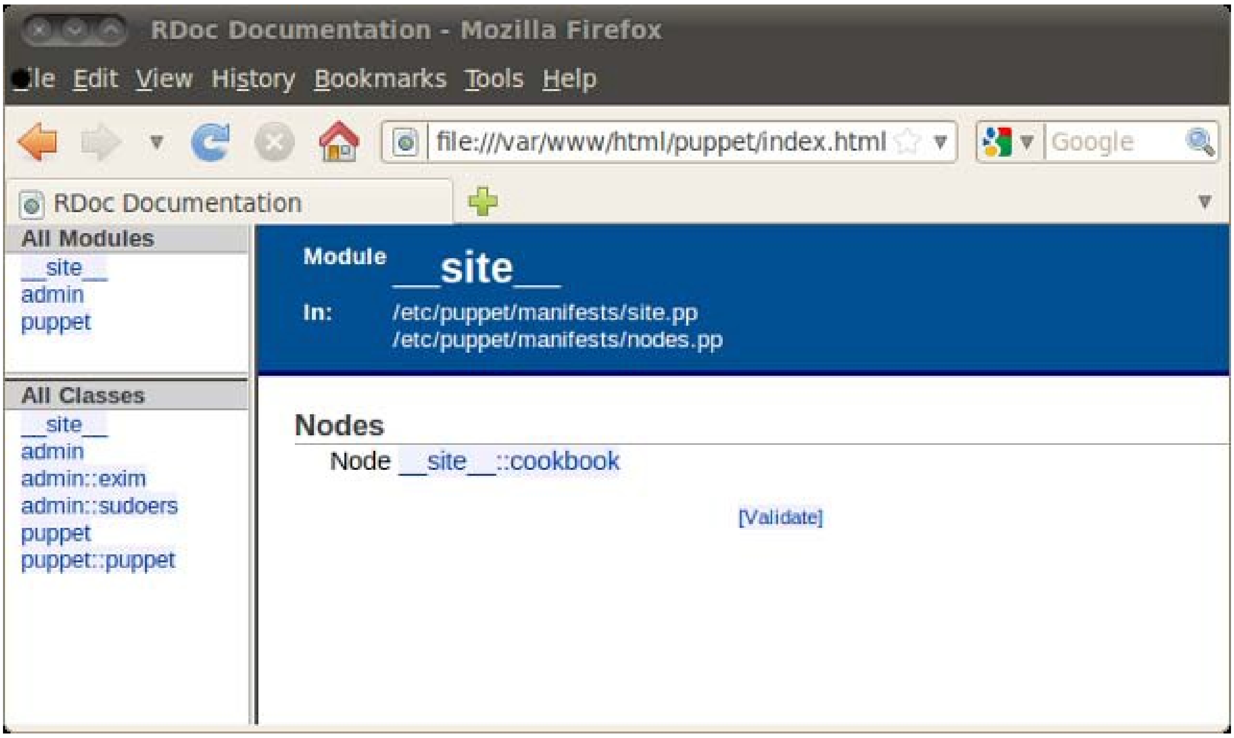
> An expert is someone who is one page ahead of you in the manual.
>
> — David Knight

像大多数工程师一样，我从来没有阅读过手册，除非或者直到产品实际出现了十万火急的情况。 然而，随着你的配置清单代码不断增多且越来越复杂，使用 Puppet 的自动文档工具 puppet doc 为你的节点（node）和类（class）生成 HTML 文档是非常有用的。

操作步骤

在你的配置清单目录下运行如下的 puppet doc 命令：

```
puppet doc --all --outputdir=/var/www/html/puppet --mode rdoc \
  --manifestdir=/etc/puppet/manifests/
```



工作原理

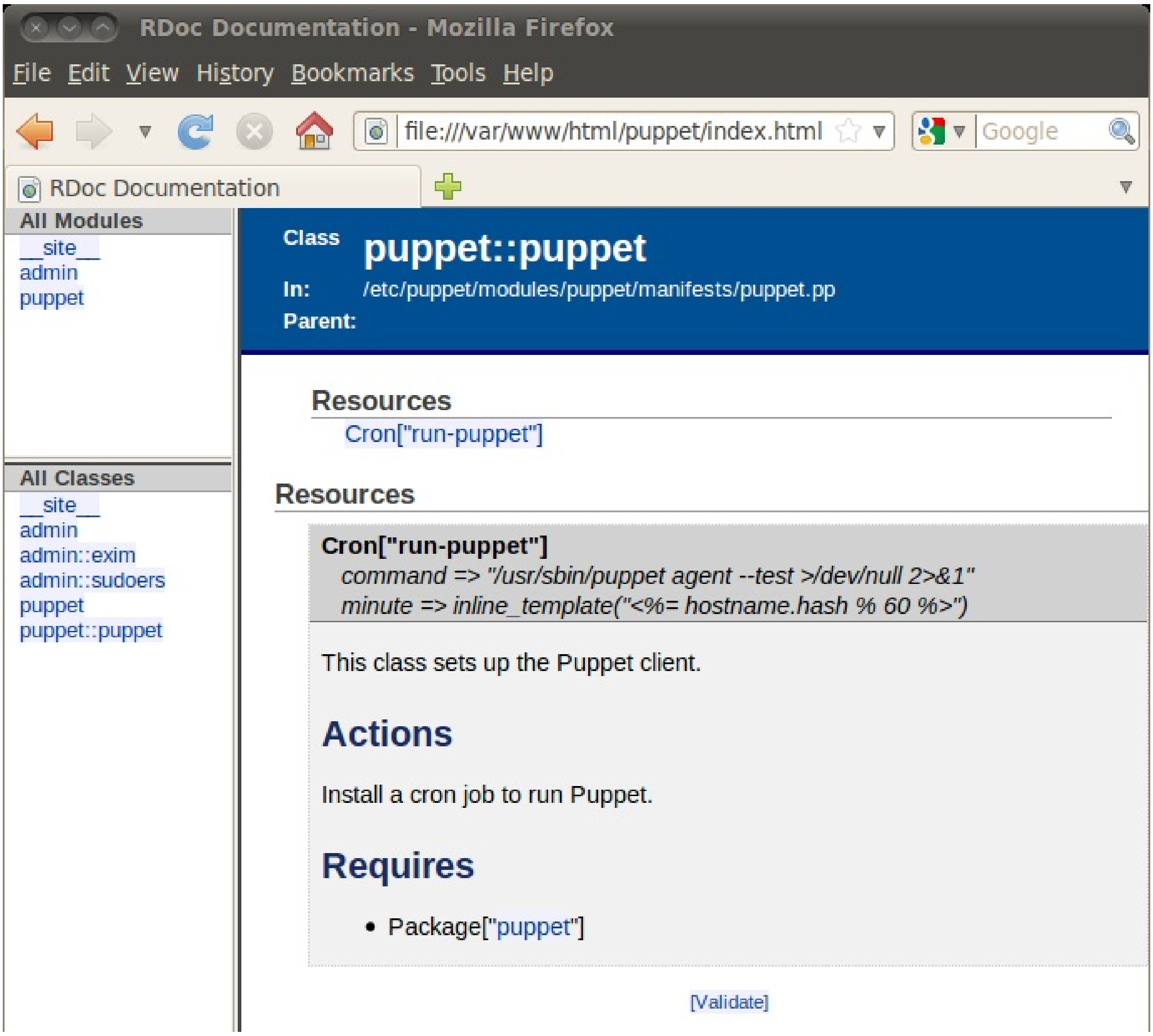
puppet doc 在 /var/www/html/puppet 目录下生成结构化的 HTML 文档树， 这与 **RDoc** 生成的文档很类似，RDoc 是流行的 Ruby 文档生成器。 这使理解不同配置清单代码之间的相互关系便得更容易， 因为你可以点击被包含的类名称便能看到它的定义。

更多用法

puppet doc 将根据你当前的配置清单生成基本的文档。 然而，你可以在你的配置清单文件中使用标准的 RDoc 语法包含更多的有用信息。 下面是一个在类中添加一些注释文档的例子：

```
class puppet {
 # This class sets up the Puppet client.
 #
 # ==Actions
 # Install a cron job to run Puppet.
 #
 # ==Requires
 # * Package["puppet"]
 #
    cron { "run-puppet":
        command => "/usr/sbin/puppet agent --test >/dev/null 2>&1",
        minute => inline_template("<%= hostname.hash.abs % 60 %>"),
    }
}
```

你在文档中为每个类添加的注释，会显示在生成的 HTML 文件里，如图所示：


RDoc Documentation - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///var/www/html/puppet/index.html
Google
RDoc Documentation
All Modules
__site__
admin
puppet
All Classes
__site__
admin
admin::exim
admin::sudoers
puppet
puppet::puppet
Class
puppet::puppet
In:
/etc/puppet/modules/puppet/manifests/puppet.pp
Parent:
Resources
Cron["run-puppet"]
Resources
Cron["run-puppet"]
command => "/usr/sbin/puppet agent --test >/dev/null 2>&1"
minute => inline_template("<%= hostname.hash % 60 %>")
This class sets up the Puppet client.
Actions
Install a cron job to run Puppet.
Requires
Package["puppet"]
[Validate]

绘制依赖关系图

依赖关系会迅速变得复杂起来，并且很容易形成 循环依赖（**circular dependency**）（即 A 依赖 B，B 又依赖 A），这将导致 Puppet 发生错误并停止工作。 幸运的是， Puppet 的 --graph 选项可以很容易生成一个资源之间的依赖关系图， 它可以帮助我们解决循环依赖的问题。

准备工作

使用如下命令安装查看图片文件所需的 graphviz 软件包：

```
# apt-get install graphviz
```

操作步骤

1. 创建 /etc/puppet/modules/admin/manifests/ntp.pp 文件，添加包含如下循环依赖的代码：

```
class admin::ntp {
    package { "ntp":
      ensure =&gt; installed,
      require =&gt; File["/etc/ntp.conf"],
    }

    service { "ntp":
      ensure =&gt; running,
      require =&gt; Package["ntp"],
    }

    file { "/etc/ntp.conf":
      source =&gt; "puppet:///modules/admin/ntp.conf",
      notify =&gt; Service["ntp"],
      require =&gt; Package["ntp"],
    }
}
```

2. 复制已存在的 ntp.conf 文件到 Puppet：

```
# cp /etc/ntp.conf /etc/puppet/modules/admin/files
```

3. 在一个节点上包含这个类：

```
node cookbook {
    include admin::ntp
}
```

4. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com

err: Could not apply complete catalog: Found 1 dependency cycle
(File[/etc/ntp.conf] =&gt; Package[ntp] =&gt; File[/etc/ntp.cor
try using the '--graph' option and open the '.dot' files in
OmniGraffle or GraphViz

notice: Finished catalog run in 0.42 seconds
```

5. 使用上面建议的 --graph 选项运行 Puppet：

```
# puppet agent --test --graph
```

6. 检查已经创建的图片文件：

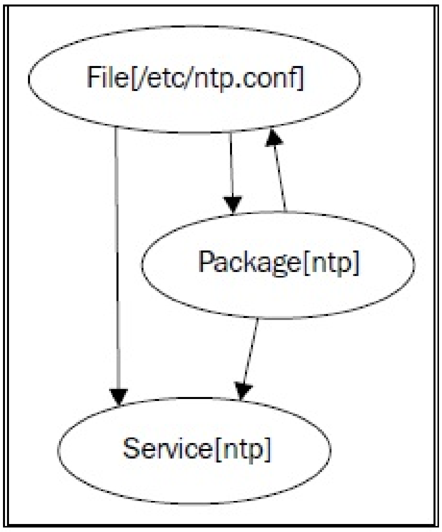
```
# ls /var/lib/puppet/state/graphs/
expanded_relationships.dot relationships.dot resources.dot
```

7. 创建一个图形化的关系依赖图

```
# dot -Tpng -o relationships.png \
  /var/lib/puppet/state/graphs/relationships.dot
```

8. 使用如下命令查看关系依赖图：

```
# eog relationships.png
```



工作原理

当你运行 puppet --graph （或者在 puppet.conf 文件中启用 graph 选项），Puppet 会生成三个 **DOT** 格式（一种图形语言）的文件。三个文件分别是：

- resources.dot：显示资源的类和层次结构，但没有依赖关系
- relationships.dot：以箭头显示资源之间的依赖关系，如上图所示
- expanded_relationships.dot：一个更详细版本的关系图

dot 工具（是 graphviz 软件包的一部分） 可以将这些文件转换为 PNG 格式进行查看。

在关系图中，你的配置清单中的每个资源显示为一个气球， 用带箭头的线连接它们表示依赖关系。 从上面的例子中我们可以看到，在 File["/etc/ntp.conf"] 和 Package["ntp"] 之间存在循环依赖关系。

为了解决循环依赖问题，你要做的就是删除一条依赖关系线，从而打破循环。

更多用法

即使你不用去寻找错误（bug），资源和其关系图也非常有用。 如果你为一个非常复杂的网络设计类和资源，研究资源关系图可以让你看起来更简单。 同样，当依赖关系变得过于复杂，通过阅读配置清单难于理解时，关系图比文档就显得更为有用。

测试你的 **Puppet** 配置清单

> If all else fails, immortality can always be assured by spectacular error.
>
> — J.K. Galbraith

你总会遭遇麻烦，就像趴在你的挡风玻璃上的一只虫子。遗憾的是像 **Nagios** 那样标准的检测监视工具也不能面面俱到地检测你要的一切。 许多技术指标对于排错是很有帮助的，例如平均负载和磁盘占用， 我更希望我的系统能提供关于应用和服务的更高级别的信息。

例如，如果你运行的是一个 web 应用程序，你不能肯定它正监听 80 端口并能返回 HTTP 200 OK 状态。也许它只是返回 Apache 默认的欢迎页面。

如果你的 web 应用程序是一个在线商店，例如：你可能希望检查以下几项：

- 大家能否看到预期的页面（例如：“欢迎光临 FooStore”）？
- 用户能否正常登录（假定应用程序支持 session 会话）？
- 搜索某种产品能否返回期待的结果？
- 响应时间是否令人满意？

这种监控（专注于应用程序的行为，而不是对服务器自身指标的度量） 经常被称作行为驱动的监控（**behavior-driven monitoring**） 。

与开发者改变代码后经常使用行为驱动的测试（behavior-driven tests） 来校验应用程序是否正常一样， 你可以在产品上线后持续地使用行为驱动的监控。

事实上，感谢有 cucumber-nagios 这样的工具存在，你可以和开发人员使用同样的测试。 Lindsay Holmwood 为流行的 **Cucumber** 测试框架所做的封装可以让你运行针对 Nagios 的基于 Cucumber 的测试，这似乎是对 Nagios 进行评估的标准方法。

准备工作

1. 为了安装 cucumber-nagios ，你首先要安装依赖的包。如果你在用 Ubuntu 或 Debian， 可能需要从源代码安装 **RubyGems** ，因为 cucumber-nagios 需要 RubyGems 1.3.6 或更高的版本。 从 RubyGems 站点： http://rubygems.org/pages/download 下载 **tarball** 。 解压缩之后运行 ruby setup.rb 编译并安装此软件包。
2. 接下来，你需要安装相关依赖包：

```
# apt-get install ruby1.8-dev libxml2-dev
```

3. 最后，使用如下命令安装 cucumber-nagios ：

```
# gem install cucumber-nagios
```

操作步骤

1. 一旦 RubyGems 和相关依赖安装完成后，你就可以开始写 Cucumber 测试了。 要做到这一点，首先使用 cucumer-nagios-gen 来帮助我们创建一个项目目录和所需要的一切：

   ```
   # cucumber-nagios-gen project mytest

   Generating with project generator:
       [ADDED] features/steps
       [ADDED] features/support
       [ADDED] .gitignore
       [ADDED] .bzrignore
       [ADDED] lib/generators/feature/%feature_name%.feature
       [ADDED] Gemfile
       [ADDED] bin/cucumber-nagios
       [ADDED] lib/generators/feature/%feature_name%_steps.rb
       [ADDED] README
   ```

   你新创建的 cucumber-nagios 项目可以在 /root/mytest 找到。

   下一步， 使用如下命令安装所需的 RubyGems：

   ```
   bundle install
   ```

   你的项目已经作为一个 git 仓库初始化了。

2. 有一个好主意是在项目目录里运行 bundle install ，因为 cucumber-nagios 建议你这么做。这样做的话 cucumber-nagios 会在项目目录里安装所有依赖的包。 然后你就可以移动项目目录到其他任何一台机器，不必再安装依赖包就可以工作。

   ```
   # cd mytest
   # bundle install
   ```

3. 现在我们开始写测试。下面的例子用于测试 Google 的主页：

   ```
   # cucumber-nagios-gen feature www.google.com home
   Generating with feature generator:
   [ADDED] features/www.google.com/home.feature
   [ADDED] features/www.google.com/steps/home_steps.rb
   ```

4. 如果你要编辑 home.feature 文件， 会发现 cucumber-nagios 已经为你生成了一个基本的测试：

```
Feature: www.google.com
  It should be up

  Scenario: Visiting home page
    When I go to "http://www.google.com"
    Then the request should succeed
```

你可以在项目下使用如下命令运行这个测试：

```
# cucumber --require features features/www.google.com/home.feat
Feature: www.google.com
  It should be up

  Scenario: Visiting home page # features/www.google.
  com/home.feature:4

    When I go to "http://www.google.com" # features/steps/http_
    steps.rb:11

    Then the request should succeed # features/steps/http_
    steps.rb:64

1 scenario (1 passed)
2 steps (2 passed)
0m0.176s
```

5. 假设一切正常（若不正常，请联系 Google）， 要对 Nagios 的检测使用此特性， 你要做的所有工作就是使用 cucumber-nagios 代替 cucumber ：

```
# bin/cucumber-nagios features/www.google.com/home.feature

CUCUMBER OK - Critical: 0, Warning: 0, 2 okay | passed=2;
failed=0; nosteps=0; total=2; time=0
```

工作原理

任何脚本都可以成为 Nagios 监控的一个插件； 它只是返回执行完成后退出的状态（0 为成功，1 为警告，2 为紧急）。cucumber-nagios 封装 Cucumber 来实现测试， 并且打印出有用信息，而后 Nagios 将通过警告或者 web 接口发送报告。

更多用法

就其本身而言，并非你在此做的所有工作都有用。但不管怎样，Cucumber 可以让你写出十分复杂的 Web 交互脚本：你可以填写表单字段、搜索、单击按扭、在页面上匹配文本等等。要监控你的 web 应用或服务的任何特性，首先弄清楚用户使用 web 浏览器的行为习惯，然后根据这些用户行为，使用 Cucumber 创建自动监控脚本。

你可以从 Cucumber 网站 http://cukes.info/ 上找到更多关于 cucumber-nagios 的信息。

执行模拟运行

> No alarms and no surprises.
>
> — Radiohead

我讨厌惊喜。有时你的 Puppet 配置清单没有像你预期的那样执行， 或者在你不知情的情况下，或许别人又提交了改变。 不管哪种情况发生，在 Puppet 执行配置清单之前能精确地获知它将要执行些什么是非常必要的。

例如，若更新了一个生产服务的配置文件并重新启动该服务，很可能会导致非计划性的停机时间。 又如，有时人为的手动编辑的配置文件会被 Puppet 同步更新时覆盖。

为了避免这些问题，你可以使用 Puppet 的 模拟运行（**dry run**）模式， 也称 无操作（**no operation**，**noop**）模式，来避免真正的执行。

操作步骤

使用 --noop 开关运行 Puppet：

```
# puppet agent --test --noop
info: Connecting to sqlite3 database: /var/lib/puppet/state/
clientconfigs.sqlite3
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1296492323'
--- /etc/exim4/exim4.conf   2011-01-17 08:13:34.349716342 -0700
+++ /tmp/puppet-file20110131-20189-127zyug-0 2011-01-31
09:45:27.792843709 -0700
@@ -1,4 +1,5 @@
 #########
+# allow spammers to use our host as a relay
 #########
notice: /Stage[main]/Admin::Exim/File[/etc/exim4/exim4.conf]/conter
{md5}02798714adc9c7bf82bf18892199971a, should be {md5}6f46256716c0
ffd6776ed059b (noop)
info: /Stage[main]/Admin::Exim/File[/etc/exim4/exim4.conf]: Schedul
refresh of Service[exim4]
notice: /Stage[main]/Admin::Exim/Service[exim4]: Would have trigger
'refresh' from 1 events
notice: Finished catalog run in 0.90 seconds
```

工作原理

在 noop 模式下，Puppet 会和常规运行模式一样运行，只是不会对客户端产生实际影响。 它会告知最终将会执行什么，你可以和你所预期的进行对比。如果有任何不同，重新检查配置清单 或机器的当前状态。

在前面的例子中，注意 Puppet 警告我们由于配置文件更新，它将会重新启动 exim 服务。 这或许是我们所期待的，也可能不是， 但不管怎样，预先知道这样的警告是有用的。 我制定了一个流程规则， 当在生产服务器上应用任何重要更新之前, 先在 noop 模式下运行 Puppet，验证发生的改变是否与我们预期一致。

更多用法

你也可以使用模拟运行模式作为一个简单的审计工具。 它会告诉你从 Puppet 最近一次应用配置清单之后的任何改变。 一些组织要求所有的配置变更都通过 Puppet 来实现，这是一种对变更实施控制的过程。 使用 Puppet 的模拟运行模式可以检测到未授权的更改，之后决定是否合并应用这些改变到 Puppet 的配置清单，或者撤消改变。

参见本书

- 第 6 章的 资源的审计 一节

检测编译错误

> My mechanic told me, *I couldn't repair your brakes, so I made your horn louder.*
>
> — Steven Wright

通常，当出现问题时，我们会在继续运行前先停止它并修复错误。然而， 当以守护进程模式运行时，Puppet 会忽略配置清单的编译错误， 仅从缓存中应用最近一次已知可运行的版本。这个行为是由 usecacheonfailure 配置设置的， 且默认值为 true：

```
# puppet --genconfig |grep usecacheonfailure
  # usecacheonfailure = true
```

值得注意的是，当你手动执行 puppet agent --test 应用配置清单时， 这种情况不会发生：如果配置清单中有任何错误，Puppet 将提示错误并拒绝执行。 这是因为 --test 开关是如下选项的缩写：

```
# puppet agent --onetime --verbose --ignorecache --no-daemonize --r
```

因为 usecacheonfailure 仅当 Puppet 以守护进程运行时有效， 有时你不会注意到一段时间内配置清单中的错误，因为 Puppet 会静默地运行旧版本的配置清单而不是显示出错信息。

操作步骤

如果你想改变这一行为，在 puppet.conf 中做如下设置：

```
    usecacheonfailure = false
```

工作原理

使用这种设置后，Puppet 将立即输出错误信息并拒绝运行，直至所有的配置清单均正确。

理解 **Puppet** 的错误信息

停一停！下面是解决错误的时间。Puppet 的错误消息会令人困惑， 有时不会包含关于如何解决问题的实质信息。

操作步骤

往往第一步是根据错误消息的文本搜索网页，看看你能找到的关于此错误的解释，并根据这些有益的建议来修复错误。 下面是一些常见的令人费解的错误以及尽可能明确的解释：

- Could not evaluate: Could not retrieve information from source(s)

  这意味着你为 source 参数指定的一个文件没有被 Puppet 发现。 请检查该文件是否存在，同时检查资源路径是否正确。

- change from absent to file failed: Could not set file on ensure: No such file or directory

  经常导致这一错误的原因是，Puppet 试图在某目录中写文件，而此目录不存在。 检查该目录是否存在或者已经在 Puppet 中被定义， 以及该目录是否为文件资源所要求的目录（这样的目录总是要首先创建的）。

- undefined method ‘closed?’ for nil:NilClass

  这种无用的错误消息相当于“出错了”。 这种错误往往可以由许多不同的原因引起， 但你可以进一步检查哪个资源、哪个类或哪个模块所引起的。 一个诀窍是添加 --debug 开关运行 puppet 从而获得更多有用的信息：

  ```
  # puppet agent --test --debug
  ```

  你也可以检查 Git 的历史日志看看最近配置清单是否有改动， 这可能是另一种方式来确定是什么扰乱了 Puppet 的正常工作。

- Could not parse for environment --- "--- production": Syntax error at end of file at line 1

  这可能是由于命令行选项输入错误引起的：例如，你输入了 puppet -verbose 而不是 puppet --verbose 。这种错误一般是很难被发现的。

- Could not request certificate: Retrieved certificate does not match private key; please remove certificate from server and regenerate it with the current key

  可能是节点的 SSL 主机密钥已经改变，或 Puppet 的 SSL 目录已被删除， 又或者你试图用与已存在的节点主机名请求新证书。 一般而言，最简单的解决方法是先从客户端删除 Puppet 的 SSL 目录（通常是 /etc/puppet/ssl） 之后在 Puppetmaster 上运行 puppet cert --clean <nodename> 。 然后再次运行 Puppet，生成正确的证书请求。

- Could not retrieve catalog from remote server: wrong header line format

  通常这种错误是在编译模板时出现的。如果在你的 ERB 上有语法错误，就会看到类似的信息， 如下面的代码片段：

```
rails_env &lt;%!= app_env %&gt;
```

- Duplicate definition: X is already defined in [file] at line Y; cannot redefine at [file] line Y

  这曾经引起我的困惑。Puppet 抱怨有重复的资源定义，通常如果你有两个同名的资源， Puppet 将会帮忙告诉你两者都定义了。 但在这种情况下，它指出了相同的文件和行号。 一个资源怎么可能是其自身的副本呢？

  答案是：若它是 define，就会出现这种情况的错误。如果你对一个 define 创建了两个实例， 并且两个实例的所有资源都包含在这个 define 中，他们需要有不同的名字。例如：

```
define check_process() {
    exec { "is-process-running?":
        command =&gt; "/bin/ps ax |/bin/grep ${name} &gt;/tmp/
 pslist.${name}.txt",
    }
}

check_process { "exim": }
check_process { "nagios": }
```

```
# puppet agent --test
info: Retrieving plugin
err: Could not retrieve catalog from remote server: Error 400 c
SERVER: Duplicate definition: Exec[is-process-running?] is alre
defined in file /etc/puppet/manifests/nodes.pp at line 22; canr
redefine at /etc/puppet/manifests/nodes.pp:22 on node cookbook.
bitfieldconsulting.com
warning: Not using cache on failed catalog
err: Could not retrieve catalog; skipping run
```

  因为 exec 资源被命名为 is-process-running。 不管你传递什么参数到 define，都会保持这个相同的名字, Puppet 会拒绝创建它的两个实例。 解决方法是在每个资源的标题部分包含实例的名字，如下所示：

```
exec { "is-process-${name}-running?":
    command =&gt; "/bin/ps ax |/bin/grep ${name} &gt;
 /tmp/pslist.${name}.txt",
}
```

显示命令的输出结果

> Computer says no.
>
> — Little Britain

一个问题的详细反馈会对解决问题有帮助。当你使用 exec 资源在节点上执行命令时， 并不总能轻易地找出其为何没有正确执行。如果命令返回一个非零的退出状态，Puppet 就会返回类似如下的错误消息：

```
err: /Stage[main]//Node[cookbook]/Exec[this-will-fail]/returns: cha
from notrun to 0 failed: /bin/ls file-that-doesnt-exist returned 2
instead of one of [0] at /etc/puppet/manifests/nodes.pp:10
```

通常我们希望看到执行失败时的输出，而不仅仅是一个退出的状态码。 你可以使用 logoutput 参数实现。

操作步骤

用 logoutput 参数定义如下的 exec 资源：

```
exec { "this-will-fail":
    command   => "/bin/ls file-that-doesnt-exist",
    logoutput => on_failure,
}
```

工作原理

现在，如果命令执行失败，Puppet 同时会打印命令的错误输出：

```
notice: /Stage[main]//Node[cookbook]/Exec[this-will-fail]/returns:
ls: cannot access file-that-doesnt-exist: No such file or directory
err: /Stage[main]//Node[cookbook]/Exec[this-will-fail]/returns: cha
from notrun to 0 failed: /bin/ls file-that-doesnt-exist returned 2
instead of one of [0] at /etc/puppet/manifests/nodes.pp:11
```

更多用法

你可以使用如下的配置为所有的 exec 资源设置打印错误输出的默认值：

```
Exec {
    logoutput => on_failure,
}
```

若不管命令执行成功与否，你都想看到其输出，需使用如下的配置：

```
    logoutput => true,
```

输出调试信息

眞理会使你获得自由。在调试问题时，若能输出配置清单某一个点上的信息将会对问题解决有很大帮助。 这是一种很好的报告问题的方式，例如，如果一个变量没有定义或者定义了一个非法的值。 有时，获知一个特定的代码片段已经被执行也是很有用的。 Puppet 的 notify 资源可以让你显示出调试信息。

操作步骤

在你的配置清单中要被调研的检查点上定义 notify 资源：

```
notify { "Got this far!": }
```

工作原理

当这个资源被编译时，Puppet 就会显示如下的信息：

```
notice: Got this far!
```

更多用法

如果你有颗勇敢的心，喜欢尝试，当然我也希望你是那样的一个人， 你或许会从自己的调试信息中发现大量的代码不能工作的原因。 所以知道如何获得更多的 Puppet 调试特性将非常有用。 下面将解释其中的一些用法。

输出变量的值

你可以在消息中引用变量：

```
notify { "operatingsystem is $operatingsystem": }
```

Puppet 会在输出中引用变量的值：

```
notice: operatingsystem is Ubuntu
```

输出资源的完整路径

对于更高级的调试， 你可以使用 withpath 参数显示哪个类的 notify 消息被执行了：

```
notify { "operatingsystem is $operatingsystem":
    withpath => true,
}
```

现在 notify 消息将显示类似如下的完整资源路径前缀：

```
notice: /Stage[main]/Nagios::Target/Notify[operatingsystem is Ubunt
message: operatingsystem is Ubuntu
```

将调试信息记录到 **Puppetmaster**

有时你仅仅想要在 Puppetmaster 上记录日志消息，而不在客户端生成额外的输出。 你可以使用 notice 函数实现：

```
notice("I am running on node $fqdn")
```

现在，当你运行 Puppet 时，就不会在客户端看到任何输出，但在 Puppetmaster 上会有一条如下的消息被记录到 syslog 系统日志：

```
Jan 31 11:51:38 cookbook puppet-master[22640]: (Scope(Node[cookbook
I am running on node cookbook.bitfieldconsulting.com
```

检查配置设置

突击测验，炙手可热！你已经知道 Puppet 的配置设置保存在 puppet.conf 文件里， 在该文件里没有提及的任何参数都会使用其默认值。 你怎样才能显示所有的配置参数的值呢（不管它是否在 puppet.conf 文件里有明确设置）？ 你可以使用 Puppet 的 --genconfig 开关。

操作步骤

运行如下命令：

```
# puppet --genconfig
```

工作原理

这将输出每个配置参数以及值（配置参数有许多）。 无论如何，输出里包含的解释每个参数的说明会非常有用。

要找到你感兴趣的特定参数，你可以像这样使用 grep 命令过滤：

```
# puppet --genconfig |grep "reportdir ="
    reportdir = /var/lib/puppet/reports
```

使用标签

标签（Tag），我们需要你！有时 Puppet 的一个类需要知道另一个类，或者至少要知道其是否已存在。 例如，一个管理防火墙的类或许需要知道一个节点是否是 web 服务器。

Puppet 的 tagged 函数会告诉你一个被命名的类或资源是否已经存在于这个节点的目录中。 你还可以对一个节点或类应用任何标签并检查这些标签是否存在。

操作步骤

1. 为了帮你辨别你是否运行在一个指定的节点或一组节点，所有节点会根据节点名被自动打标签， 并从父节点那里继承名称。

```
node bitfield_server {
    include bitfield
}

node cookbook inherits bitfield_server {
    if tagged("cookbook") {
        notify { "this will succeed": }
    }
    if tagged("bitfield_server") {
        notify { "so will this": }
    }
}
```

2. 为了告诉你一个节点是否包含一个指定的类， 所有的节点会用他们包含的所有类其父类的名字自动打标签：

```
include apache::port8000

if tagged("apache::port8000") {
    notify { "this will succeed": }
}

if tagged("apache") {
    notify { "so will this": }
}
```

3. 如果你想要在一个节点上设置指定的标签，则需使用 tag 函数：

```
tag("old-slow-server")
if tagged("old-slow-server") {
    notify { "this will succeed": }
}
```

4. 如果你想要对一个指定的资源设置标签，则需使用 tag 元参数（**metaparameter**）：

```
file { "/etc/ssh/sshd_config":
    source =&gt; "puppet:///modules/admin/sshd_config",
    notify =&gt; Service["ssh"],
    tag    =&gt; "security",
}
```

5. 你也可以使用标签来确定哪一部分配置清单将被应用。 如果你在 Puppet 的命令行上使用 --tags 选项，仅仅被打上了指定标签的类或资源被应用。 例如，你只想要更新 exim 的配置而不想运行其他部分的配置清单，执行如下命令：

```
# puppet agent --test --tags exim
```

更多用法

你可以使用标签创建一个资源的集合。例如，如果某服务需要依赖一大批文件片段， 你可以使用下面的配置：

```
class firewall::service {
    service { "firewall":
        …
    }

    File <| tag == "firewall-snippet" |> ~> Service["firewall"]
}

class myapp {
    file { "/etc/firewall.d/myapp.conf":
        tag => "firewall-snippet",
        …
    }
}
```

在此，我们指定如果更新了被标记为 firewall-snippet 的任何 file 资源， 就通知 firewall 服务。 我们所要做的全部工作就是为特定的应用程序或服务添加防火墙配置片段， 且应用程序或服务是打了 firewall-snippet 标签的， 剩下的工作交由 Puppet 完成。

虽然我们添加了 notify => Service["firewall"] 到每一个资源片段，但是如果 我们定义的 firewall 服务不断变化，就不得不捕获这些变化并据此更新所有的片段。 标签可以让我们把相应的代码进行逻辑封装，这样也使得将来的维护和重构会更加容易。

使用运行阶段

> What do you get when you play country music backwards? You get your girl back, your dog back, your pick-up back, and you stop drinking.
>
> — Louis Saaberda

重要的是要以正确的顺序做事情。 一个普遍的需求是应用一个特定的资源之前必须先应用它所依赖的所有资源 （例如，安装一个软件包仓库），或者之后运行其他资源 （例如，一旦其所依赖的包都已安装，就部署应用程序）。 Puppet 的 **run stages** 用于实现这种功能。

操作步骤

1. 添加如下代码到你的配置清单：

```
class install_repos {
    notify { "This will be done first": }
}

class deploy_app {
    notify { "This will be done last": }
}

stage { "first": before =&gt; Stage["main"] }
stage { "last": require =&gt; Stage["main"] }

class { "install_repos": stage =&gt; "first" }
class { "deploy_app": stage =&gt; "last" }
```

2. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303127505'
notice: This will be done first
notice: /Stage[first]/Beginning/Notify[This will be done first]
message: defined 'message' as 'This will be done first'
notice: This will be done last
notice: /Stage[last]/End/Notify[This will be done last]/message
defined 'message' as 'This will be done last'
notice: Finished catalog run in 0.59 seconds
```

工作原理

1. 首先把要做的工作根据先后顺序声明为两个类。

```
class install_repos {
    notify { "This will be done first": }
}

class deploy_app {
    notify { "This will be done last": }
}
```

2. 然后创建一个名为 first 的运行阶段：

```
stage { "first": before =&gt; Stage["main"] }
```

参数 before 指定了 first 阶段必须在 main 阶段（默认阶段）之前完成的一切。

3. 接着创建一个名为 last 的运行阶段：

```
stage { "last": require =&gt; Stage["main"] }
```

参数 require 指定了在 last 阶段之前 main 阶段必须完成的一切。

4. 最后，我们引用 install_repos 和 deploy_app 两个类， 指定他们 分别应该是 first 阶段和 last 阶段的一部分：

```
class { "install_repos": stage =&gt; "first" }
class { "deploy_app": stage =&gt; "last" }
```

注意我们使用的是 class 关键字，而不是 include 关键字， 就像为类传递参数一样。 你能想象，作为一个参数，stage 可以被传递到任何类。

5. Puppet 会按照如下的顺序应用 stage：

```
i. first
ii. main
iii. last
```

更多用法

事实上，只要你喜欢你可以定义许多运行阶段，并为他们安排执行顺序。 这可以简化配置清单的复杂性，否则在很大程度上就会需要很多资源之间的明确的依赖关系。 如果你可以将所有资源分为A和B两组，A组资源必须要在B组资源之前运行完成， 这是使用运行阶段的典型情况。

Gary Larizza 写了一篇有用的关于使用运行阶段的介绍文章，并结合了一些实际的例子， 网址为 http://glarizza.posterous.com/using-run-stages-with-puppet 。

使用不同的环境

> A Zen student went up to a hot dog vendor and said: "Make me one with everything".
>
> — Joke

环境背景很重要。 如果你想对 Puppet 的配置清单在应用到生产环境之前先进行测试， 你可以使用 Puppet 的 **environment** 特性来做到。 这可以让你根据环境应用不同的配置清单来设置客户机。例如，你可以定义如下的环境：

- development
- staging
- production

你可以在 puppet.conf 文件中设置环境。在本例中，我们将添加一个 development 环境，用于指向一套不同的配置清单。

操作步骤

在 puppet.conf 文件中添加如下行：

```
[development]
manifest = /etc/puppet/env/development/manifests/site.pp
modulepath = /etc/puppet/env/development/modules:/etc/puppet/module
```

工作原理

你可以根据你的喜好将一套与一个环境相关的配置清单存在磁盘的任何位置， 只要设置 manifest 参数指向顶级的 site.pp 文件即可。在本例中, 我们为 development 环境所编制的配置清单保存在 /etc/puppet/env/development 目录。 类似地，你还需要使用 modulepath 参数指定这个环境需要使用的模块目录。

在上面的例子中，参数 modulepath 同时包括了 /etc/puppet/modules； 所以若 Puppet 在你的 development 环境中找不到模块，它也会在默认的环境中寻找。 这就意味着你仅需要将区别于其他环境的模块存入 development 环境。

默认的环境是 production，所以如果你没有指定 Puppet 的环境的话，将会使用默认环境。

更多用法

如果你使用 Git 这样的版本控制系统，你的不同环境可以是 Git 的不同分支。 一旦你编制了新模块并完成了测试，就可以将其合并（merge）到用于生产环境的 Git 主分支（master branch）。 你可以在 R.I. Pienaar 的文章里了解更多使用环境的技巧： http://www.devco.net/archives/2009/10/10/puppet_environments.php 。

> 译者注
>
> 你还可以参考 http://puppetlabs.com/blog/git-workflow-and-puppet-environments/ 了解关于使用 Git 和 Puppet 动态环境的技巧。

你可以用多种方式指定客户端的环境。 一种方式是在运行 Puppet 时使用 --environment 开关：

```
# puppet agent --test --environment=development
```

另一种方式是在客户端的 puppet.conf 文件中使用 environment 参数指定：

```
[main]
environment=development
```

如果你使用的是外部节点分类器脚本（将在本书第 9 章的描述），也可以指定客户端所属的环境。

你也可以为每一个环境指定不同的 fileserver.conf（参考 配置 Puppet 的文件服务器 一节）。 为了实现此功能，需要在 Puppetmaster 的配置文件 puppet.conf 中为每个环境设置 fileserverconfig 参数：

```
[development]
fileserverconfig = /etc/puppet/fileserver.conf.development

[production]
fileserverconfig = /etc/puppet/fileserver.conf.production
```

更多的信息，请参考 Puppet Labs 的“使用多环境”的页面：http://projects.puppetlabs.com/projects/1/wiki/Using_Multiple_Environments 。

参见本书

- 第 1 章的 使用版本控制 一节
- 第 3 章的 使用模块 一节
- 第 9 章的 使用外部节点分类器 一节

Puppet 语言及其写作风格

> Computer language design is just like a stroll in the park. Jurassic Park, that is.
>
> — Larry Wall

在本章中，我们将学习如下内容：

- 使用 Puppet 社区规范
- 使用模块
- 使用标准的命名规范
- 使用嵌入式 Ruby 代码
- 使用纯 Ruby 代码书写配置清单
- 遍历多个项目
- 书写强大的条件语句
- 在 if 语句中使用正则表达式
- 使用选择器和 case 语句
- 检测字符串中是否包含指定的值
- 使用正则表达式替换

> Elegance is not a dispensable luxury, but a factor that decides between success and failure.
>
> — Edsger W. Dijkstra

在本章，你将学习如何书写优雅的 Puppet 配置清单。“优雅”的含义体现在几个方面： 可读性、高效性、书写符合社区规范的代码。

我们将会看到，如何遵循社区规范将你的代码组织并构造成模块，以便他人能很容易地阅读和维护你的代码。 我还会向你展示一些 Puppet 语言的强大功能，使你写出更简洁而传神的配置清单。

使用 **Puppet** 社区规范

> A society made up of individuals who were all capable of original thought would probably be unendurable.
>
> — H. L. Mencken

“随大溜” 有时会是个好主意。如果其他人需要阅读或管理你的配置清单， 或者如果你想要分享代码到社区， 那么尽可能遵循现有的样式约定会是个好主意。

操作步骤

1. 将你的资源名用双引号括起来；例如使用 package { "exim4": 而不是 package { exim4:

   当使用一些像连字符和空格这样的字符时很容易引起 Puppet 解析器的混乱，明智而又安全的做法是始终使用双引号将所有的资源名括起来。

2. 总是将非 Puppet 保留字的参数值用双引号括起来，例如：

   ```
   name  =&gt; "Nucky Thompson",
   mode  =&gt; "0700",
   owner =&gt; "deploy",
   ```

   但保留字不用使用引号：

   ```
   ensure =&gt; installed,
   enable =&gt; true,
   ensure =&gt; running,
   ```

   当在字符串中引用变量的值时，始终使用大括号将变量名括起来。例如：

   ```
   source =&gt; "puppet:///modules/webserver/${brand}.conf",
   ```

   否则 Puppet 的解析器不得不猜测哪些字符是变量名的一部分。使用大括号将使其变得清晰。

3. 总是以逗号结束参数声明的行，即使它是最后一个参数：

   ```
   service { "memcached":
       ensure =&gt; running,
       enable =&gt; true,
   }
   ```

很多时候，当你编辑文件时会追加一个额外的参数， 而忘记在原来最后一行的末尾添加必要的逗号！

4. 当声明仅有一个参数的资源时，将声明放在一行上且无需使用结尾的逗号：

```
package { "puppet": ensure =&gt; installed }
```

5. 当声明有多个参数的资源时，每个参数占一行：

```
package { "rake":
    ensure   =&gt; installed,
    provider =&gt; gem,
    require  =&gt; Package["rubygems"],
}
```

6. 当声明 符号链接（**symlinks**）时，使用 ensure => link：

```
file { "/etc/php5/cli/php.ini":
    ensure =&gt; link,
    target =&gt; "/etc/php.ini",
}
```

7. 为了使代码便于阅读，所有的箭头以最长一个参数对齐，例如：

```
file { "/var/www/${app}/shared/config/rvmrc":
    owner   =&gt; "deploy",
    group   =&gt; "deploy",
    content =&gt; template("rails/rvmrc"),
    require =&gt; File["/var/www/${app}/shared/config"],
}
```

每个资源中的箭头（=>）都要对齐，而不是每个文件中的箭头都对齐； 否则会在不同文件之间复制代码时带来麻烦。

更多用法

Puppet Labs 网站上维护着 Puppet 社区规范指南文档：http://docs.puppetlabs.com/guides/style_guide.html 。

Tim Sharpe 写了一个 puppet-lint 工具， 你可以用它检查你的配置清单是否符合规范。 运行 gem install puppet-lint 安装它，详细的信息请参考 https://github.com/rodjek/puppet-lint 。

使用模块

你会对自己编写的 Puppet 代码感到羞愧吗？其他人看你的代码时会有恐惧感吗？为了使你的 Puppet 配置清单更清晰且易于维护，一件最重要的事情就是将这些配置清单组织成 模块（**modules**）。

模块是对相关事物进行分组的一种简单方式；例如，一个 webserver 模块可能包含作为一个 Web 服务器所需的一切，包括 Apache 配置文件，虚拟主机配置模板以及部署这些所需的 Puppet 代码。

将配置清单拆分成模块易于代码的重用和共享；也是一种组织配置清单的最合乎逻辑的方式。 在本节中，我们将会创建一个管理 memcached 的模块，memcached 是一个 Web 应用程序常用的内存缓存系统。

操作步骤

1. 在 puppet.conf 中查找你的模块路径，其默认值是 /etc/puppet/modules。 如果你像我前面建议的那样将 Puppet 配置清单纳入了版本控制， 那么就使用你的工作副本中的 modules 目录，之后再由他部署到 /etc/puppet/modules。

   ```
   # puppet --genconfig |grep modulepath
   modulepath = /etc/puppet/modules:/usr/share/puppet/modules
   ```

   在模块路径下创建一个名为 memcached 的子目录：

   ```
   # cd /etc/puppet/modules
   # mkdir memcached
   ```

   在此目录中，创建 manifests 和 files 两个子目录：

   ```
   # cd memcached
   # mkdir manifests files
   ```

   在 manifests 目录下，以如下内容创建 init.pp 文件：

```
class memcached {
    package { "memcached":
        ensure =&gt; installed,
    }

    file { "/etc/memcached.conf":
        source =&gt; "puppet:///modules/memcached/memcached.cor
    }

    service { "memcached":
        ensure  =&gt; running,
        enable  =&gt; true,
        require =&gt; [ Package["memcached"],
                     File["/etc/memcached.conf"] ],
    }
}
```

切换到 files 目录，以如下内容创建 memcached.conf 文件：

```
-m 64
-p 11211
-u nobody
-l 127.0.0.1
```

2. 为了使用你的新模块，你的节点定义中添加如下的代码：

```
node cookbook {
    include memcached
}
```

3. 运行 Puppet 检测新配置：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1300361964'
notice: /Stage[main]/Memcached/Package[memcached]/ensure: ensur
changed 'purged' to 'present'
...
info: /Stage[main]/Memcached/File[/etc/memcached.conf]:
Filebucketed /etc/memcached.conf to puppet with sum a977521922a
c959ac953712840803
notice: /Stage[main]/Memcached/File[/etc/memcached.conf]/conter
content changed '{md5}a977521922a151c959ac953712840803' to '{mc
f5c0bb01a24a5b3b86926c7b067ea6ba'
notice: Finished catalog run in 20.68 seconds
```

4. 检查新配置的服务是否正在运行：

```
# service memcached status
  * memcached is running
```

工作原理

模块有特定的目录结构。并非所有子目录都要存在，但如果存在就该以如下的布局方式组织：

```
MODULEPATH/
 `-- MODULE_NAME
          |-- files/
          |-- templates/
          |-- manifests/
          |--   ...
          `-- README
```

在 memcached.pp 文件中定义了 memcached 类， 它可以被 Puppet 自动导入。现在将其包含到节点中：

```
include memcached
```

在 memcached 类中，引用了 memcached.conf 文件：

```
file { "/etc/memcached.conf":
    source => "puppet:///modules/memcached/memcached.conf",
}
```

正如我们在 配置 Puppet 的文件服务器 一节讲到的， 上面代码中的 source 参数告诉 Puppet 在如下的路径寻找文件：

```
MODULEPATH/
    memcached/
        files/
            memcached.conf
```

更多用法

学习喜欢模块的组织方式，使用模块将使你的 Puppet 管理生活便得更轻松。模块并不复杂。 然而，实践和经验会帮助你判断何时应该将事物划分并组织成模块，以及如何更好的安排你的模块结构。 如下所述的一些技巧会对你有所帮助。

模板

模板作为模块的一部分，如果你需要使用它，那么其放在 MODULE_NAME/templates 目录，参考如下的用法：

```
file { "/etc/memcached.conf":
    content => template("memcached/memcached.conf"),
}
```

Puppet 会在如下目录搜索模板文件：

```
MODULEPATH/
    memcached/
        templates/
            memcached.conf
```

Fact、函数、类型和提供者

模板中也可以包含自定义 fact、自定义函数、自定义资源类型和自定义提供者。 关于这些内容的详细信息请参阅 外部工具和 Puppet 的生态系统 一章的内容。

puppet-module

你也可以使用 puppet-module 工具为一个新模块创建目录布局，而不是使用手工方式。 请参考第 9 章的 使用公共模块 一节获得更详细的信息。

第三方模块

你可以下载由其他人开发的模块，并在你的配置清单中使用这些模块， 就像这些模块是你自己写的一样。 请参考第 9 章的 使用公共模块 一节获得更详细的信息。

模块的组织方法

有关如何组织模块的更多信息，参见 Puppet Labs 站点：http://docs.puppetlabs.com/guides/modules.html 。

参见本书

- 第 1 章的 配置 Puppet 的文件服务器 一节
- 第 9 章的 创建 Facter 的自定义 fact 一节
- 第 9 章的 使用公共模块 一节
- 第 9 章的 创建自定义的资源类型 一节
- 第 9 章的 创建自定义的提供者 一节

使用标准的命名规范

> There are only two hard problems in computer science: cache invalidation, naming things, and off-by-one errors.
>
> — Phil Karlton

当你维护代码时会发现，选择适当而有意义的信息为你的模块或类命名是有很大帮助的。 尤其是当其他人需要阅读或基于你编写的配置清单工作时，这种命名方法是十分必要的。

操作步骤

1. 模块应该以被管理的软件或服务命名，例如：apache 或 haproxy。

   用跟在模块之后的功能或提供的服务为模块中的类命名，例如：apache::vhosts 或 rails::dependencies。

   如果在模块中提供了禁用服务的功能，将其命名为 disabled。例如一个禁用 Apache 的类应该命令为 apache::disabled。

2. 如果一个节点需要提供多种服务，请在节点的定义中包含一个类或者每个服务的类名。例如：

```
node server014 inherits server {
    include puppet::server
    include mail::server
    include repo::gem
    include repo::apt
    include zabbix
}
```

3. 管理用户的模块应该命名为 user。

   i. 在 user 模块中，声明虚拟用户的类应命名为 user::virtual。

   ii. 在 user 模块中，一个特定的用户组应该命名为子类，例如 user::sysadmins 或 user::contractors。

4. 当你需要覆盖某些特定节点或服务的类时，继承该类并以节点名作为子类名的前缀。 例如，如果节点 cartman 需要一个特殊的 SSH 配置且要覆盖 ssh 类，可以这样做：

```
class cartman_ssh inherits ssh {
    [ override config here ]
}
```

5. 当使用 Puppet 为不同的服务部署配置时，被 Puppet 解析后传出的文件应该以服务开头， 随后跟上指示文件功能的后缀。例如：

   - Apache init 脚本?—?apache.init
   - Rails 的日志滚动配置片段?—?rails.logrotate
   - Nginx 为 mywizzoapp 应用所做的 vhost 配置?—?mywizzoapp.vhost.nginx
   - 独立运行的 MySQL 服务器的配置?—?standalone.mysql

   如果你要管理不同的 Ruby 版本，可以使用添加了版本号的类名，表示由此类负责特定版本的管理， 例如 ruby192 或 ruby186。

更多用法

Puppet 社区维护着一套 Puppet 基础设施的最佳实践准则，其中包括对命名的提示： http://projects.puppetlabs.com/projects/1/wiki/Puppet_Best_Practice2 。

一些人更愿意在一个节点中使用逗号分割的列表包含多个类，而不是使用分离的 include 语句。例如：

```
node server014 inherits server {
    include puppet::server,
            mail::server,
            repo::gem,
            repo::apt,
            zabbix
}
```

这是个风格问题，但我更愿意使用分离的 include 语句，一行包含一个模块或类，因为这样可以更容易地在节点之间复制或移动，而无需每次都要整理逗号和缩进的问题。

我在前面的例子中曾几次提到过继承。如果你不知道这是什么，不要担心：我会在下一章详细解释。

使用嵌入式 Ruby 代码

> Ruby, like fire, is a very useful friend, and a very dangerous enemy.
>
> — Mikkel Bruun

在模板中使用嵌入式 Ruby 帮助构建动态的配置文件或实现数组遍历是一种强大的方式。 然而，你也可以在配置清单中使用 inline_template 函数直接嵌入 Ruby 而不必使用分离的模板文件。

操作步骤

在 Puppet 的配置清单中将 Ruby 代码传递给 inline_template 函数：

```
cron { "nightly-job":
    command => "/usr/local/bin/nightly-job",
    hour => "0",
    minute => inline_template("<%= hostname.hash.abs % 60 %>"),
}
```

工作原理

传递给 inline_template 的字符串都会被执行，就像 ERB 模板一样。 也就是说，在分隔符 <%= 和 %> 之间的所有内容都被当做 Ruby 代码执行， 其余的将被视为字符串。

参见本书

- 第 5 章的 使用 ERB 模板 一节
- 第 5 章的 在模板中遍历数组 一节

使用纯 Ruby 代码书写配置清单

你会说西班牙语吗？学习一门语言可能会很有趣，但并非所有人都想这么做。Puppet 有时会因为它使用自己专用的配置语言而不是现存的通用语言（如 Ruby）书写配置清单而遭到批评。

不是每个人都认为这是个缺点。计算机科学家 Dennis Ritchie 说：

> A language that doesn’t have everything is actually easier to program in than some that do.
>
> — Dennis Ritchie

无论你是什么观点，这种批评已不再适用。因为 Puppet 已经拥有了使用 Ruby 语言书写配置清单的实验性支持，这在生产环境上是相当有用的，即使还处于早期开发阶段。 在你的配置清单中可以混合使用两种语言，Puppet 以文件扩展名进行区别，扩展名 .rb 表示用 Ruby 语言书写的配置清单文件，而扩展名 .pp 表示 Puppet 专用的配置语言文件。

为了书写配置清单使用的 Ruby 的 特定领域语言（**domain-specific language，DSL**） 看上去与标准的 Puppet 语言非常相似。 在下面的例子中，我将向你展示如何将典型的 Puppet 配置清单转换成 Ruby 的。 Puppet 语法的原始配置清单如下：

```
class admin::exim {
    package { "exim4": ensure => installed }

    service { "exim4":
        ensure  => running,
        require => Package["exim4"],
    }

    file { "/etc/exim4/exim4.conf":
        content => template("admin/exim4.conf"),
        notify  => Service["exim4"],
        require => Package["exim4"],
    }
}
```

操作步骤

1. 使用如下内容创建 /etc/puppet/modules/admin/manifests/exim.rb 文件：

```
hostclass "admin::exim" do
    package "exim4", :ensure =&gt; :installed

    service "exim4",
        :ensure  =&gt; :running,
        :require =&gt; "Package[exim4]"

    file "/etc/exim4/exim4.conf",
        :content =&gt; template(["admin/exim4.conf"]),
        :notify  =&gt; "Service[exim4]",
        :require =&gt; "Package[exim4]"
end
```

2. 在一个节点中包含这个类并运行 Puppet。

工作原理

1. 关键字 hostclass 声明一个类，就像 Puppet 中的 class：hostclass admin::exim do
2. 然后跟一个 do ... end 语句块，这相当于 Puppet 的一对大括号。
3. 通过在资源类型后调用函数来声明资源： 例如 package 或 service：package "exim4", :ensure => :installed
4. 传递给函数的参数是一个用逗号分割的列表，参数必须要用双引号括起来， 或者在参数前使用前导的冒号使其成为 Ruby symbol 对象： :ensure => :running,

   又如，像 :installed 或 :running 这样的 Puppet 内置名字都是 Ruby 的 symbol 对象。

   | | 译者注

   有关 Ruby Symbol 的详细介绍请参考： http://www.ibm.com/developerworks/cn/opensource/os-cn-rubysbl/

   |

5. 当我们需要引用资源来表示相互关系时，就要使用 :require， 资源标识符通过首字母大写的资源类型跟上写在一对方括号中的资源名给出，例如： require => "Package[exim4]"

   在调用像模板这样的函数时可以使用函数名跟一对圆括号，在圆括号中传递的参数是 一对用方括号声明的数组，例如：:content => template(["admin/exim4.conf"]),。

更多用法

Ruby DSL 尚处于早期开发阶段。试验虽然有趣，除非你有使用 Ruby 的令人信服的理由否则先别用它， 至今我仍旧使用标准的 Puppet 语言。或许将来 Ruby DSL 会被广泛应用，但在此期间， 你会发现没有它的生活会更轻松。然而，若你坚持要使用它，下面介绍几个非常有用的提示。

变量

当你像常规的 Ruby 程序一样使用 Ruby 变量时，可以使用 scope.lookupvar 访问你的 Puppet 变量，例如：

```
notice( "I am running on node %s" % scope.lookupvar("fqdn") )
```

将得到：

```
notice: I am running on node cookbook.bitfieldconsulting.com
```

要在你的 Puppet 配置清单范围内设置变量，使用 scope.setvar，例如：

```
require 'time'
scope.setvar("now", Time.now)
notice( "At the third stroke, the time sponsored by Bitfield
Consulting will be: %s" % scope.lookupvar("now") )
```

上面的代码将获得如下结果：

```
notice: At the third stroke, the time sponsored by Bitfield Consult
will be: Wed Mar 23 05:58:16 -0600 2011
```

文档

你可以在 Puppet Labs 网站找到更多如何使用 Ruby DSL 的详细资料，包括诸如 **virtual resources** 和 **collections** 这样的高级主题：http://projects.puppetlabs.com/projects/1/wiki/Ruby_Dsl 。

Ken Barber 提供了一些语法例子，并对 Puppet 语法和 Ruby DSL 结构做了比较，网址在 https://github.com/bobsh/puppet-rubydsl-examples 。

最后，James Turnbull 发布过一篇 blog， 展示了使用 Ruby 连接 MySQL 服务器的高级方法： http://www.puppetlabs.com/blog/using-ruby-inthe-puppet-ruby-dsl/ 。

遍历多个项目

该死的东西一个接一个！ 数组（**Arrays**）是 Puppet 的一个强大特性； 不论何时，你要对列表中的元素执行相同的操作，数组就会帮你的忙。 你可以创建一个数组，将所有的数组元素放在一对方括号中并以逗号间隔，例如：

```
$lunch = [ "franks", "beans", "mustard" ]
```

操作步骤

在你的配置清单中添加如下代码：

```
$packages = [ "ruby1.8-dev",
              "ruby1.8",
              "ri1.8",
              "rdoc1.8",
              "irb1.8",
              "libreadline-ruby1.8",
              "libruby1.8",
              "libopenssl-ruby" ]

package { $packages: ensure => installed }
```

运行 Puppet，值得注意的是现在每个软件包都应该被安装。

工作原理

当 Puppet 遇到数组作为资源名的情况时，它会对数组中的每个元素创建一个资源。 在前面的例子中，对 $packages 数组中的每一个包， 使用相同的参数（ensure => installed）创建了一个新的 package 资源。 这是对很多类似的资源进行实例化的一个非常紧凑的方式。

更多用法

如果你听到哈希（hash），会比数组更兴奋。

哈希

哈希（**hash**）与数组类似，但它的每个元素都可以通过名字存储和查找。例如：

```
$interface = { name    => 'eth0',
               address => '192.168.0.1' }

notice("Interface ${interface[name]} has address ${interface[addres
```

的执行结果为：

```
Interface eth0 has address 192.168.0.1
```

你可以给哈希赋任意的值：字符串、函数调用、表达式、甚至其它哈希或数组。

使用 **split** 函数创建数组

你可以使用方括号来声明文字数组，例如：

```
define lunchprint() {
    notify { "Lunch included $name": }
}

$lunch = [ "egg", "beans", "chips" ]
lunchprint { $lunch: }
```

执行结果为：

```
Lunch included egg
Lunch included beans
Lunch included chips
```

但是 Puppet 还可以使用 split 函数从一个字符串创建数组，例如：

```
$menu = "egg beans chips"
$items = split($menu, ' ')
lunchprint { $items: }
```

执行结果为：

```
Lunch included egg
Lunch included beans
Lunch included chips
```

注意 split 函数携带两个参数：第一个参数是要被拆分的字符串；第二个参数是拆分间隔符， 在本例中是一个空格。当 Puppet 遍历字符串时，一遇到空格就将其视为一个元素的结束和下一个元素的开始。 所以，给定的字符串 "egg beans chips" 将被拆分为三个元素。

拆分间隔符可以是任意字符或字符串：

```
$menu = "egg and beans and chips"
$items = split($menu, ' and ')
```

拆分间隔符也可以是正则表达式（regular expression），例如：一个多选一集合可以使用 | (pipe) 符号来做间隔：

```
$lunch = "egg:beans,chips"
$items = split($lunch, ':|,')
```

书写强大的条件语句

生活充满选择。Puppet 的 if 语句允许基于变量或表达式的值应用不同的配置代码。 使用 if 语句，你可以根据当前节点的实际情况应用不同的资源或参数值，例如： 操作系统或内存大小。举例来说，数据中心A中的节点和数据中心B中的节点可能需要使用不同的 DNS 服务器；对 Ubuntu 系统和其他系统可能需要执行不同类的集合。

操作步骤

在你的配置清单中添加如下的代码：

```
if $lsbdistid == "Ubuntu" {
    notice("Running on Ubuntu")
} else {
    notice("Non-Ubuntu system detected. Please upgrade to Ubuntu
immediately.")
}
```

工作原理

Puppet 对 if 关键字之后的内容视为表达式并对其进行评估。若表达式的值为 true， Puppet 将执行大括号内的代码。

另外，你还可以使用 else 分支，它将在表达式的值为 false 时执行。

更多用法

虽然你可以在 Puppet 中写出很复杂的 if 语句，但是我不建议你这么做。 很多时候，改变你的设计思路（例如，使用模板）比使用 if 语句要好。

从我用于生产环境的一些配置清单实例来看，我惊讶地发现，在所有的几千行代码中都没有使用 if 语句。尽管如此，你的里程可能会有所不同，所以下面给出一些使用 if 的技巧。

elsif

你可以添加 elsif 关键字进行进一步的测试，例如：

```
if $lsbdistid == "Ubuntu" {
    notice("Running on Ubuntu")
} elsif $lsbdistid == "Debian" {
    notice("Close enough…")
} else {
    notice("Non-Ubuntu system detected. Please upgrade to Ubuntu immediately.")
}
```

比较

你可以使用 == 语法检查两个值是否相同，例如：

```
if $lsbdistid == "Ubuntu" {
    ...
}
```

或者使用 != 语法检查两个值是否不同，例如：

```
if $lsbdistid != "CentOS" {
    ...
}
```

你也可以使用 < 和 > 比较两个数值，例如：

```
if $uptime_days > 365 {
    notice("Really .. there have been kernel security patches out there for ages, you will so be 0wned!")
}
```

还可以使用 <= 或 >= 对数值进行比较：

```
if $lsbmajdistrelease <= 9 {
    ...
}
```

复杂表达式

你可以把上述代码片段中的一些简单表达式组合成更复杂的逻辑表达式， 逻辑表达式使用与（and）、或（or）和非（not）连接简单表达式，例如：

```
if ($uptime_days > 365) and ($lsbdistid == "Ubuntu") {
     ...
}

if ($role == "webserver") and ( ($datacenter == "A") or ($datacente
     ...
}
```

参见本书

- 第 3 章的 在 if 语句中使用正则表达式 一节
- 第 3 章的 检测字符串中是否包含指定的值 一节
- 第 3 章的 使用选择器和 case 语句 一节

在 if 语句中使用正则表达式

> Some people, when confronted with a problem think; *I know, I'll use regular expressions.* Now they have two problems.
>
> — Jamie Zawinski

你可以在 if 语句中使用另一种类型的表达式，即 正则表达式（**regular expression**）。 正则表达式是一种使用模式匹配的强大的字符串比较方式。

操作步骤

在你的配置清单中添加如下代码：

```
if $lsbdistdescription =~ /LTS/ {
    notice("Looks like you are using a Long Term Support version of
 Ubuntu.")
} else {
    notice("You might want to upgrade to a Long Term Support versio
 of Ubuntu...")
}
```

工作原理

Puppet 将两个斜线之间的文本当做正则表达式对待，两个各斜线之间的文本就是要匹配的内容。 如果正则表达式匹配成功，if 表达式为真，第一个大括号之间的代码就会被执行。

若要使用匹配的反逻辑，即不匹配，需要使用 !~ 替代 =~ ，例如：

```
if $lsbdistdescription !~ /LTS/ {
```

更多用法

正如 Jamie Zawinski 指出的，正则表达式虽然强大，但难于理解和调试。 如果你发现自己所写的正则表达式相当复杂，以至于一眼看上去不能被理解， 就应该考虑简化设计使正则表达式更易懂。 然而，正则表达式的一个特别有用的功能是能够捕获模式。

捕捉模式

你不仅可以使用正则表达式匹配文本，还可以捕获匹配的文本并将其存储在变量中：

```
$input = "Puppet is better than manual configuration"
if $input =~ /(.*) is better than (.*)/ {
    notice("You said '$0'. Looks like you're comparing $1 to $2!")
}
```

结果为：

```
You said 'Puppet is better than manual configuration'. Looks like y
comparing Puppet to manual configuration!
```

变量 $0 存储了所有匹配的文本（假设整体匹配成功）。如果你将一部分正则表达式置于一对圆括号中，就会创建一个组（**group**），并且所有被匹配的组都会存储在变量中。第一个被匹配的组是 $1，第二个是 $2，以此类推，正如上例展示的那样。

正则表达式语法

Puppet 使用了 Ruby 正则表达式语法（Regular expression syntax）的一个子集，如果你还不熟悉正则表达式，下面网址中的解释会对你有用：http://gnosis.cx/publish/programming/regular_expressions.html 。

参见本书

- 本章的 使用正则表达式替换 一节

使用选择器和 case 语句

> Smarts is the most exclusive club in town. Everyone welcome.
>
> — Sign

有时选择性是很重要的。尽管你可以使用 if 书写任何条件语句， 但 Puppet 还提供了帮助你更容易地表达条件的额外形式，例如： 选择器（selector）和 case 语句。

操作步骤

1. 在你的配置清单中添加如下代码：

```
$systemtype = $operatingsystem ? {
    "Ubuntu" =&gt; "debianlike",
    "Debian" =&gt; "debianlike",
    "RedHat" =&gt; "redhatlike",
    "Fedora" =&gt; "redhatlike",
    "CentOS" =&gt; "redhatlike",
    default  =&gt; "redhatlike",
}

notify { "You have a ${systemtype} system": }
```

2. 接下来，再添加如下代码：

```
class debianlike {
    notify { "Special manifest for Debian-like systems": }
}

class redhatlike {
    notify { "Special manifest for RedHat-like systems": }
}

case $operatingsystem {
    "Ubuntu",
    "Debian": {
        include debianlike
    }
    "RedHat",
    "Fedora",
    "CentOS": {
        include redhatlike
    }
}
```

工作原理

上面的例子同时演示了选择器（selector）和 case 语句，让我们来看看它们是如何工作的。

- **selector**

  在第一个例子中，我们使用了一个选择器（操作符 ?）根据变量 $operatingsystem 的值为 $systemtype 变量赋值。 这类似于 C 语言和 Ruby 语言中的三元运算符， 不同之处在于三元运算符只能在两个可选值之间选择，而此处的选择器则可以提供你想要的多个值。

  Puppet 会与 $operatingsystem 的每一个可能的值（如 Ubuntu、Debian等）一一做比较。 这些值可以是正则表达式（例如，部分字符串匹配，或使用通配符），但在本例中我们仅仅使用了文本字符串。 一旦发现匹配的值，选择器表达式就会返回与其相关的匹配字符串。 例如，如果 $operatingsystem 的值是 Fedora，选择器表达式就会返回字符串 redhatlike 并将其赋予变量 $systemtype。

- **Case** 语句

  与选择器不同，case 语句不会返回任何值。 case 语句适用于， 当你想根据一个表达式的不同的值执行不同代码的情况。 在我们的第二个例子中，使用 case 语句根据 $operatingsystem 的值或者包含 debianlike 类，或者包含 redhatlike 类。

  再次指出，Puppet 会根据 $operatingsystem 值与潜在的匹配列表进行比较。这些匹配列表可以是正则表达式，或者是字符串，或者像我们的例子中一样使用 逗号间隔的字符串列表。 当 Puppet 在匹配列表中找到一个匹配值，就会执行与此匹配项相关的大括号之间的代码。所以， 如果 $operatingsystem 的值是 Ubuntu，那么 include debianlike 就会被执行。

更多用法

一旦你掌握了选择器和 case 语句的基本用法，你会发现下面的技巧非常有用。

正则表达式

与 if 语句一样，你可以在选择器和 case 语句中使用正则表达式，并且你也可以 捕获匹配组的值，并使用 $1、$2 等引用它们的值。

```
case $lsbdistdescription {
    /Ubuntu (.+)/: {
        notify { "You have Ubuntu version $1": }
    }
    /CentOS (.+)/: {
        notify { "You have CentOS version $1": }
    }
}
```

默认值

选择器和 case 语句都可以让你指定一个 default 值，当没有其他匹配项时就使用这个默认值：

```
$lunch = "Sausage and chips"
$lunchtype = $lunch ? {
    /chips/ => "unhealthy",
    /salad/ => "healthy",
    default => "unknown",
}

notify { "Your lunch was ${lunchtype}": }
```

结果为：

```
Your lunch was unhealthy
```

检测字符串中是否包含指定的值

想知道什么在什么不在吗？Puppet 的 in 可以帮你， 如下面的表达式：

```
if "foo" in $bar
```

如果 foo 是 $bar 的子串，表达式的值为 true。 如果 $bar 是个数组，并且 foo 是这个数组中的一个元素，表达式的值为 true。 如果 $bar 是一个哈希，foo 是 $bar 的一个键值，表达式的值为 true。

操作步骤

1. 在你的配置清单中添加如下代码：

```
if $operatingsystem in [ "Ubuntu", "Debian" ] {
    notify { "Debian-type operating system detected": }
} elsif $operatingsystem in [ "RedHat", "Fedora", "SuSE", "Cent
] {
    notify { "RedHat-type operating system detected": }
} else {
    notify { "Some other operating system detected": }
}
```

2. 运行 Puppet：

```
# puppet agent --test
Debian-type operating system detected
```

更多用法

in 表达式既可以使用在 if 语句或其他条件语句中，还可以在任何表达式能出现的地方使用。 例如，你可以像下面这样为一个变量赋值：

```
$debianlike = $operatingsystem in [ "Debian", "Ubuntu" ]

if $debianlike {
    $ntpservice = "ntp"
} else {
    $ntpservice = "ntpd"
}
```

使用正则表达式替换

> Change is inevitable, except from vending machines.
>
> — Robert C. Gallagher

Puppet 的 regsubst 函数提供了一种处理文本的简单方法， 用于在字符串中查找和替换，或者从字符串提取匹配的模式。 例如，我们通常需要对从 facter 或者从外部程序获得的数据做这样的处理。

在本例中将会看到如何使用 regsubst 提取一个 IP 地址的前三个字节 （即网络地址部分，假定此 IP 地址是一个 C 类地址）。

操作步骤

1. 在配置清单中添加如下的代码：

    ```
    $class_c = regsubst($ipaddress, "(.*)\..*", "\1.0")
    notify { $ipaddress: }
    notify { $class_c: }
    ```

2. 运行 Puppet：

    ```
    notice: 10.0.2.15
    notice: 10.0.2.0
    ```

工作原理

regsubst 函数需要携带三个参数： 源字符串、匹配模式（pattern）和替换结果。 在本例中，我们指定的源字符串是 $ipaddress，这恰好是：

```
10.0.2.15
```

我们还指定了 **(.*)\..*** 作为匹配模式并且 \1.0 作为替换结果。

匹配模式将匹配整个 IP 地址，捕获的前三个字节放在一对圆括号内。 被捕获的文本可以在替换结果中使用 \1 来引用。

被匹配的全部文本（本例中是字符串 “10.0.2.15”）将使用替换结果（replacement）来替换。 这里是 \1 （从源字符串中捕获的文本）跟上字符串 .0，最终获得：10.0.2.0。

更多用法

匹配模式中可以使用任何正则表达式，与在 if 语句中使用的正则表达式 (Ruby) 语法一致。

参见本书

- 第 4 章的 导入动态信息 一节
- 第 4 章的 获得系统的环境信息 一节
- 第 3 章的 在 if 语句中使用正则表达式 一节

书写更优质的配置清单

> There are only two kinds of programming languages: those people always bitch about and those nobody uses.
>
> — Bjarne Stroustrup

在本章中，我们将学习如下内容：

- 使用资源的数组
- 使用 define 资源
- 指定资源的依赖关系
- 使用节点继承
- 使用类的继承和重载
- 给类传递参数
- 书写可重用的跨平台配置清单
- 获得系统的环境信息
- 导入动态信息
- 从 CSV 文件导入数据
- 给 Shell 命令传递参数

你的 Puppet 配置清单实际上就是你所管理的整个基础设施的活生生的文档。 保持配置清单的整洁和良好组织使其更容易维护和理解是个明智之举。 Puppet 为你方便管理代码提供了许多工具，其中包括：

- 数组（Arrays）
- 定义（Defines）
- 依赖（Dependencies）
- 继承（Inheritance）
- 类参数（Class parameters）

我们将看到如何使用上面的所有工具（甚至更多）。通过对本章的阅读，尝试使用这些例子做实验， 看看利用这些功能是否能帮你简化和改善你的 Puppet 代码。

使用资源的数组

你可以将一切均视为 Puppet 的资源，你也可以使用数组资源。 使用这种想法，可以重构你的配置清单，使其更加简洁而清晰。

操作步骤

1. 在你的配置清单中，将同一类型资源的几个实例定义在一个类中，例如：packages：

```
package { "sudo" : ensure =&gt; installed }
package { "unzip" : ensure =&gt; installed }
package { "locate" : ensure =&gt; installed }
package { "lsof" : ensure =&gt; installed }
package { "cron" : ensure =&gt; installed }
package { "rubygems" : ensure =&gt; installed }
```

2. 使用如下的方法用数组将它们组织在一起并用一个 package 资源代替：

```
package { [ "cron",
            "locate",
            "lsof",
            "rubygems"
            "screen",
            "sudo"
            "unzip" ]:
    ensure =&gt; installed,
}
```

工作原理

绝大多数的 Puppet 资源类型都可以使用数组来替换单个的资源名， 并会为每个数组的元素创建一个实例。 你为资源提供的所有参数（例如：ensure => installed）都会分配给每个新的资源实例。

参见本书

- 第 3 章的 遍历多个项目 一节

使用 define 资源

> Girl number twenty unable to define a horse!" said Mr. Gradgrind.?—?Charles Dickens.
>
> — Hard Times

除非你知道如何定义你想要什么，不然不会得到预期的结果。 在上一节的示例中，我们看到了如何将同类资源组合成数组从而减少冗余代码。 然而，这种技术有一个限制，那就是所有的资源必须使用相同的参数。 当你有一组资源拥有一些公用的参数而其中一些资源确有不同的参数时， 就需要使用 define 资源将它们组合在一起。

操作步骤

1. 在配置清单中添加如下代码：

```
define tmpfile() {
    file { "/tmp/$name":
        content =&gt; "Hello, world",
    }
}

tmpfile { ["a", "b", "c"]: }
```

2. 运行 Puppet：

```
notice: /Stage[main]//Node[cookbook]/Tmpfile[a]/File[/tmp/a]/
ensure: defined content as '{md5}bc6e6f16b8a077ef5fbc8d59d0b931
notice: /Stage[main]//Node[cookbook]/Tmpfile[b]/File[/tmp/b]/
ensure: defined content as '{md5}bc6e6f16b8a077ef5fbc8d59d0b931
notice: /Stage[main]//Node[cookbook]/Tmpfile[c]/File[/tmp/c]/
ensure: defined content as '{md5}bc6e6f16b8a077ef5fbc8d59d0b931
```

工作原理

你可以认为 **define** 就像是一个饼干切割器。它描述了一种模式，Puppet 可以用它创建许多类似的资源。任何时候你都可以在你的配置清单中声明 tmpfile 实例，Puppet 将会插入包含在 tmpfile 定义中的所有资源。

在我们的例子中，名为 tmpfile 的 define 包含了一个 file 资源， 其 content 为 "Hello, world"；其 path 为 "/tmp/$name"。 如果你用名字 foo 声明了一个 tmpfile 实例，如下所示：

```
tmpfile { "foo": }
```

会被 Puppet 要创建的实例名字所代替。这就像是我们创建了一个新的资源类型：tmpfile，它包含一个参数（即其名字）。

与常规资源一样，我们不仅可以为其传递一个字符串的名字，我们还可以传递一个数组名， Puppet 会对每一个数组元素创建一个 tmpfile 实例，就像上面的例子那样。

更多用法

在上面的例子中，我们定义的 define 仅有一个名字参数，不同实例的名字不同。但是我们可以为其添加任何我们想要的参数，只要我们在 define 中声明这些参数即可：

```
define tmpfile( $greeting ) {
    file { "/tmp/$name":
        content => $greeting,
    }
}
```

当我们声明一个资源的实例时，可以为其传递参数值，例如：

```
tmpfile{ "foo": greeting => "Hello, world" }
```

你可以使用逗号间隔的列表同时声明多个参数：

```
define webapp( $domain, $path, $platform ) {
    ...
}

webapp { "mywizzoapp":
    domain   => "mywizzoapp.com",
    path     => "/var/www/apps/mywizzoapp",
    platform => "Rails",
}
```

这是对某些常见的资源进行抽象的一项强大的技术，抽象出资源之间的共性并保存在一个地方， 你就不必每次都做重复劳动（**Don't Repeat Yourself**）。 在上面的例子中，webapp 里或许有许多独立的资源：软件包（packages）、配置文件（config files）、 源码检出（source code checkouts)、虚拟主机（virtual hosts）等等， 但是除了我们为 webapp 提供的参数不同之外，所有实例要执行的工作都相同。 这些可能会在模板中引用，例如：为一个虚拟主机设置域名。

指定资源的依赖关系

> Remove wrapper, open mouth, insert muffin, eat.
>
> — Instructions on 7-11 muffin packaging

为确保事物以正确的顺序发生，你可以在 Puppet 中指定一个资源依赖另一个资源， 例如：你必须先安装软件包 X 然后再启动它提供的服务，因此应该标记这项服务依赖于软件包 X。 Puppet 会按要求的顺序排出它遇到的所有依赖。

在一些配置管理系统中，资源按照你的书写顺序被应用，换句话说，资源被应用的顺序是隐式的。 Puppet 则不是这种情况，资源或多或少以一种随机（但一致）顺序被应用， 除非你明确地使用依赖关系指出应用的顺序。 一些人更喜欢隐式的方式，因为你可以按照你需要的执行顺序书写资源定义， 并且这就是它们被执行的方式。

另一方面，许多情况下资源的顺序无关紧要。使用隐式风格的系统时， 你不能明确地告诉系统，资源 B 是否被写在了资源 A 之后，由于资源 B 依赖于 A； 或者说书写的顺序是否正确，即资源 A 写在了资源 B 之前。 这使得重构更加困难，因为移动资源有可能会打破一些隐式的依赖关系。

虽然 Puppet 会让你多做一点儿工作，就是预先指定依赖关系， 但是这样产生的代码会更清晰且更易于维护。让我们看一个例子。

操作步骤

1. 使用如下的内容创建一个新文件 /etc/puppet/modules/admin/manifests/ntp.pp：

```
class admin::ntp {
    package { "ntp":
        ensure =&gt; installed,
    }

    service { "ntp":
        ensure  =&gt; running,
        require =&gt; Package["ntp"],
    }

    file { "/etc/ntp.conf":
        source  =&gt; "puppet:///modules/admin/ntp.conf",
        notify  =&gt; Service["ntp"],
        require =&gt; Package["ntp"],
    }
}
```

2. 复制已经存在的 ntp.conf 文件到 Puppet 的如下目录：

```
# cp /etc/ntp.conf /etc/puppet/modules/admin/files
```

3. 在 nodes.pp 中将 admin::ntp 类添加到你的服务器：

```
node cookbook {
    include admin::ntp
}
```

4. 现在删除系统中已存在的 ntp.conf 文件：

```
# rm /etc/ntp.conf
```

5. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1302960655'
notice: /Stage[main]/Admin::Ntp/File[/etc/ntp.conf]/ensure:
defined content as '{md5}3386aaad98dd5e0b28428966dac9e1f5'
info: /Stage[main]/Admin::Ntp/File[/etc/ntp.conf]: Scheduling
refresh of Service[ntp]
notice: /Stage[main]/Admin::Ntp/Service[ntp]: Triggered 'refres
from 1 events
notice: Finished catalog run in 2.36 seconds
```

工作原理

在本例中演示了两种类型的依赖：require 和 notify。在第一种情况中，ntp 服务要求 ntp 包资源首先被应用：

```
service { "ntp":
    ensure  => running,
    require => Package["ntp"],
}
```

在第二种情况中，NTP 的配置文件设置成了 notify（通知）ntp 服务；换句话说，一旦发现配置文件有变化，Puppet 就应该使用新的配置文件重新启动 ntp 服务：

```
file { "/etc/ntp.conf":
    source  => "puppet:///modules/admin/ntp.conf",
    notify  => Service["ntp"],
    require => Package["ntp"],
}
```

这意味着服务依赖于配置文件以及所安装的软件包，Puppet 会按照如下的正确顺序来应用这三个资源：

```
Package["ntp"] -> File["/etc/ntp.conf"] ~> Service["ntp"]
```

事实上，这是指定相同依赖关系链的另一种方法。添加上面这行到你的配置清单中， 就会产生和上例中使用 require 和 notify 参数的同样效果 （-> 表示 require，~> 表示 notify）。然而，我更喜欢使用 require 和 notify， 因为依赖关系被定义成了资源的一部分，因此更容易看清将会发生什么。 不过，对于复杂的依赖关系链，你可能想使用 **->** 符号来代替。

更多用法

你也可以指定一个资源依赖于某个类：

```
require => Class["my-apt-repo"]
```

你不仅可以指定资源和类之间的依赖关系，甚至可以指定 **collections** 之间的依赖关系：

```
Yumrepo <| |> -> Package <| provider == yum |>
```

这是一种功能强大的表达方式，所有 provider 是 yum 的 package 资源被应用之前， 所有的 yumrepo 资源首先都应该被应用。

> 历史说明:
>
> 在 Puppet 2.7 版本之前， 所有资源编目都以非确定性的方式被应用， 这意味着每次 Puppet 运行资源的顺序都会不同。这可能会导致一些有趣的问题， 比如一个 Puppet 配置清单在一台机器上能运行成功而在另一台机器上却运行失败。 经过 Puppet Labs 的努力，现在这种情况已经不会发生。现在 Puppet 既保证可靠成功，又保证可靠失败（"either succeed reliably, or fail reliably"）。如果你还在使用早期版本并且遇到了这种问题，请更新到新版本。

使用节点继承

将所有服务器都放在一个篮子里的系统管理员是勇敢的（或者说是愚蠢的）。 比方说，你将服务器托管在三个不同的提供商：WreckSpace、GoDodgy 和 VerySlow。 他们有不同的数据中心且分布在不同的地理位置，所以你需要对服务器配置做一些小的改动， 以适应每个提供商。你有几种类型的服务器，它们随机分布在三个不同的提供商。

使用 Puppet 的一种实现方法是在节点定义中设置一个变量用于告知每个节点处于哪个提供商：

```
node webserver127 {
    $provider = "VerySlow"
    include admin::basics
    include admin::ssh
    include admin::ntp
    include puppet::client
    include backup::client
    include webserver
}

node loadbalancer5 {
    $provider = "WreckSpace"
    include admin::basics
    include admin::ssh
    include admin::ntp
    include puppet::client
    include backup::client
    include loadbalancer
}
```

正如你看到的，结果中存在大量的重复行。而使用下面的方法会比上面的方法容易得多。例如： 首先简单地定义一种 WreckSpace 节点，然后使用从这个节点 继承（**inherit**） 的方法创建新节点，在新节点中只需要包含它要执行的类：比如 loadbalancer 或 webserver 即可。

操作步骤

1. 为所有的节点创建一个基类，它仅包含每个节点都要执行的类，例如：

```
node server {
    include admin::basics
    include admin::ssh
    include admin::ntp
    include puppet::client
    include backup::client
}
```

2. 通过这个 server 节点创建三个不同的子类，每个子类都设置了适当的 provider 变量：

```
node wreckspace_server inherits server {
    $provider = "WreckSpace"
}

node gododgy_server inherits server {
    $provider = "GoDodgy"
}

node veryslow_server inherits server {
    $provider = "VerySlow"
}
```

3. 现在，为了要在 VerySlow 创建一个新的 Web 服务器，仅需要从 veryslow_server 继承即可：

```
node webserver904 inherits veryslow_server {
    include webserver
}
```

工作原理

当一个节点继承自另一个节点，它会应用父节点的所有配置。 然后你可以添加任何代码，从而使得这个节点成为有别于其他节点的特殊节点。

你可以配置一个节点继承自另外一个节点，而另外一个节点也可以继承自其它节点等。 但是你不能继承自多个节点（即不能多重继承），因此不能使用如下方式定义节点：

```
node movable_server inherits gododgy_server, veryslow_server,
wreckspace_server {
 # This won't work
}
```

更多用法

与定义一个普通节点一样，你可以指定一个节点列表，这些节点都继承自相同的节点定义：

```
node webserver1, webserver2, webserver3 inherits wreckspace_server
    ...
}
```

另外，你也可以使用正则表达式匹配多个服务器节点：

```
node /webserver\d+.veryslow.com/ inherits veryslow_server {
    ...
}
```

参见本书

- 本章的 使用类的继承和重载 一节

使用类的继承和重载

正如节点可以从其他节点继承一样，这可以为相似的节点复制很多代码， 同样的思想也可以用于类。

例如，假设你有一个管理 Apache Web 服务器的 apache 类， 你想要使用略有不同的配置文件设置一台新的 Apache 机器?—?也许是监听的端口不同。

你可以复制整个 apache 类，除了配置文件。另外，你可以从 apache 类中提取配置文件并创建两个新类，每个新类都包含 apache 基类并添加一个新版本的配置文件。

一个更简洁的做法是从 apache 类继承，而后仅覆盖其配置文件。

准备工作

1. 为新的 apache 模块创建目录结构：

```
# mkdir /etc/puppet/modules/apache
# mkdir /etc/puppet/modules/apache/manifests
# mkdir /etc/puppet/modules/apache/files
```

2. 使用如下代码创建 /etc/puppet/modules/apache/manifests/init.pp 文件：

```
class apache {
    package { "apache2-mpm-worker": ensure =&gt; installed }

    service { "apache2":
        enable  =&gt; true,
        ensure  =&gt; running,
        require =&gt; Package["apache2-mpm-worker"],
    }

    file { "/etc/apache2/ports.conf":
        source =&gt; "puppet:///modules/apache/port80.conf.apac
        notify =&gt; Service["apache2"],
    }
}
```

3. 若 Apache 软件包还未安装，安装它，复制其包含的文件 ports.conf 到 Puppet：

```
# apt-get install apache2-mpm-worker

# cp /etc/apache2/ports.conf \
/etc/puppet/modules/apache/files/port80.conf.apache
```

4. 添加 apache 类到一个节点，例如：

```
node cookbook {
    include apache
}
```

5. 运行 Puppet 验证配置清单是否正常工作。

操作步骤

1. 创建 port80.conf.apache 文件的一个新版本 port8000.conf.apache，并做如下改动：

```
NameVirtualHost *:8000
Listen 8000
```

2. 使用如下内容创建一个新文件 /etc/puppet/modules/apache/manifests/port8000.pp：

```
class apache::port8000 inherits apache {
    File["/etc/apache2/ports.conf"] {
        source =&gt; "puppet:///modules/apache/port8000.conf.ap
    }
}
```

3. 改变你的节点配置，使其包含 apache::port8000 类而不是 apache 类：

```
node cookbook {
    include apache::port8000
}
```

4. 运行 Puppet 检查它是否会按照要求的那样发生改变：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1302970905'
--- /etc/apache2/ports.conf 2010-11-18 14:16:23.000000000 -070
+++ /tmp/puppet-file20110416-6165-pzeivi-0 2011-04-16
10:21:47.204294334 -0600
@@ -5,8 +5,8 @@
# Debian etch). See /usr/share/doc/apache2.2-common/NEWS.Debia
# README.Debian.gz

-NameVirtualHost *:80
-Listen 80
+NameVirtualHost *:8000
+Listen 8000

&lt;IfModule mod_ssl.c&gt;
# If you add NameVirtualHost *:443 here, you will also have
to change
info: FileBucket adding /etc/apache2/ports.conf as {md5}38b31d2
6f3640a8dfbe1ff5d1c4ad
info: /Stage[main]/Apache/File[/etc/apache2/ports.conf]:
Filebucketed /etc/apache2/ports.conf to puppet with sum
38b31d20326f3640a8dfbe1ff5d1c4ad
notice: /Stage[main]/Apache/File[/etc/apache2/ports.conf]/conte
content changed '{md5}38b31d20326f3640a8dfbe1ff5d1c4ad' to '{md
1d9d446f779c55f13c5fe5a7477d943'
info: /Stage[main]/Apache/File[/etc/apache2/ports.conf]:
Scheduling refresh of Service[apache2]
notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refres
from 1 events
notice: Finished catalog run in 4.85 seconds
```

工作原理

让我们再看看这个新类：

```
class apache::port8000 inherits apache {
    File["/etc/apache2/ports.conf"] {
        source => "puppet:///modules/apache/port8000.conf.apache",
    }
}
```

你可以从类名后看出，此类继承（inherits）自 apache 类。这将创建一个与 apache 类完全相同的副本，除了跟随其后的变化。

如下的代码片段：

```
File["/etc/apache2/ports.conf"] {
```

指定了我们想要改变父类中名为 /etc/apache2/ports.conf 的 file 资源（注意 File 是首字母大写的，这意味着，我们指的是现有的资源，而不是定义一个新资源）。

如下的代码片段：

```
source => "puppet:///modules/apache/port8000.conf.apache",
```

意味着我们将使用一个新的值覆盖父类中 source 资源的参数值。 如果我们复制整个 apache 类的定义并改变资源 source 的值，那么结果将是完全一样的：

```
class apache {
    package { "apache2-mpm-worker": ensure => installed }

    service { "apache2":
        enable  => true,
        ensure  => running,
        require => Package["apache2-mpm-worker"],
    }

    file { "/etc/apache2/ports.conf":
        source => "puppet:///modules/apache/port8000.conf.apache",
        notify => Service["apache2"],
    }
}
```

更多用法

首先覆盖被继承的类看上去有些复杂。然而一旦你掌握了这种思想，就会发现这实际上很简单。 这是一种使你的配置清单更具可读性的强大方式，因为这样消除了大量的重复代码， 使你仅专注于编写不同的代码部分。下面给出几种使用覆盖的方法。

取消参数的定义

有时候你不想改变一个参数的值，只是想完全移除它的值。 为了实现这一点，可以使用 undef 值覆盖原有值。 其结果就像是此参数从未在先前定义过一样。

```
class apache::norestart inherits apache {
    File["/etc/apache2/ports.conf"] {
        notify => undef,
    }
}
```

使用 **+>** 操作符添加额外的值

与替换一个值类似，你可能想要在父类定义的基础上添加更多的值。 使用 **plusignment** 操作符 **+>** 可以实现这一功能：

```
class apache::ssl inherits apache {
    file { "/etc/ssl/certs/cookbook.pem":
        source => "puppet:///modules/apache/cookbook.pem",
    }

    Service["apache2"] {
        require +> File["/etc/ssl/certs/cookbook.pem"],
    }
}
```

操作符 **+>** 在父类定义的值的基础上添加一个值（或使用方括号括起来的一个数组）。 对于上面的例子，我们最终得到的代码相当于：

```
service { "apache2":
    enable  => true,
    ensure  => running,
    require => [ Package["apache2-mpm-worker"], File["/etc/ssl/cert
 cookbook.pem"] ],
}
```

禁用资源

继承和覆盖最常见的用途之一就是禁用服务或其他资源：

```
class apache::disabled inherits apache {
    Service["apache2"] {
        enable => false,
        ensure => stopped,
    }
}
```

参见本书

- 第 4 章的 使用节点继承 一节
- 第 4 章的 给类传递参数 一节
- 第 3 章的 使用标准的命名规范 一节

给类传递参数

有时对一个类的某些方面进行 参数化（**parameterize**）是很有用的。例如， 你可能需要管理不同版本的 gem 软件包，既可以为每一种版本创建分离的单独的类，也可以使用继承和覆盖，为一个类传递一个版本号作为参数。

操作步骤

1. 声明参数作为如下类定义的一部分：

```
class eventmachine( $version ) {
    package { "eventmachine":
        provider =&gt; gem,
        ensure   =&gt; $version,
    }
}
```

2. 然后在一个节点上使用如下语法包含类：

```
class { "eventmachine": version =&gt; "0.12.8" }
```

工作原理

类定义：

```
class eventmachine( $version ) {
```

与普通的类定义一样，不同之处在于它携带了一个参数：$version。 在此类中，我们定义了如下的 package 资源：

```
package { "eventmachine":
    provider => gem,
    ensure   => $version,
}
```

这是一个 gem 包资源，并且要求安装 $version 版本的软件包。

当你在一个节点上包含这个类时， 替换通常的语法：

```
include eventmachine
```

为如下的 class 语句：

```
class { "eventmachine": version => "0.12.8" }
```

这有相同的效果，只是同时为参数 $version 指定了一个值。

更多用法

你可以为一个类指定多个参数：

```
class mysql( $package, $socket, $port ) {
```

使用同样的方法为其传递参数值：

```
class { "mysql":
    package => "percona-sql-server-5.0",
    socket  => "/var/run/mysqld/mysqld.sock",
    port    => "3306",
}
```

你也可以为一些参数指定默认值：

```
class mysql( $package, $socket, $port = "3306" ) {
```

或者使用如下代码片段为所有参数指定默认值：

```
class mysql(
    package = "percona-sql-server-5.0",
    socket = "/var/run/mysqld/mysqld.sock",
    port = "3306" ) {
```

与 define 不同，一个节点上只能存在一个参数化的类实例。 所以当你需要针对一个资源创建多个不同的实例时，应该使用 define 取代类的参数化。

参见本书

- 本章的 使用节点继承 一节
- 本章的 使用类的继承和重载 一节

书写可重用的跨平台配置清单

每个系统管理员都梦想着使用统一的、同质的基础设施，相同的机器都运行相同的操作系统且其版本相同。 正如生活中的其他领域一样，现实往往是凌乱的、往往与理想情况不符。

你很可能会负责管理一堆使用年限不同、架构不同、运行不同发行版本的不同内核的服务器， 常常是分散在不同的数据中心和不同的互联网服务供应商（ISP）。

在这种情况下，会造成系统管理员心中对在 for 循环中执行 SSH（"SSH in a for loop"）的恐惧， 因为在每个服务器上即使执行相同的命令，也可能产生不同的、不可预知的、甚至是危险的结果。

当然，我们应该努力将旧服务器进行更新，并将一切尽可能工作在一个单一的参考平台之上， 从而使管理更简单，更廉价，更可靠。但在达到这一目标之前，我们可以使用 Puppet， 它可以使我们更容易地应对异构环境（heterogeneous environments）。

操作步骤

1. 如果你有一些放置在不同数据中心的服务器，这些服务器需要略有不同的网络配置， 例如，使用节点继承技术来封装差异：

```
node wreckspace_server inherits server {
        include admin::wreckspace_specific
}
```

2. 你需要应用相同的配置清单到运行着不同 OS 发行版本的服务器，其重要差别可能在于软件包名、 服务名以及配置文件的存放位置。可以通过在一个类中使用选择器（selectors） 捕获这些差异并设置全局变量的值：

```
$ssh_service = $operatingsystem? {
     /Ubuntu|Debian/ =&gt; "ssh",
     default =&gt; "sshd",
}
```

之后你就不用担心配置清单其他部分的差异了，当你要提及这些时， 可以放心的使用变量，它会根据具体环境正确的指向相应的正确值：

```
service { $ssh_service:
    ensure =&gt; running,
}
```

3. 我们经常需要配合不同的架构；这可能会影响共享库的路径，也可能需要不同版本的软件包。 同样地，尝试在一个单一的 architecture 类中封装所需要的全局变量的值：

```
$libdir = $architecture ? {
    x86_64 =&gt; "/usr/lib64",
    default =&gt; "/usr/lib",
}
```

之后在需要一个架构相关的值时，你就可以引用这些全局变量的值， 无论是在配置清单中引用还是在模板中引用均可：

```
; php.ini
[PHP]
; Directory in which the loadable extensions (modules) reside.
extension_dir = &lt;%= libdir %&gt;/php/modules
```

工作原理

这种方法的优点（可以称为“自上而下”）是你仅需要进行一次选择。 另一种是自下而上的方法，使用这种方法，在全部配置清单中你随处可见 使用 selector 或 case 语句的设置：

```
service { $operatingsystem? {
    /Ubuntu|Debian/ => "ssh",
    default         => "sshd" }:
    ensure          => running,
}
```

这不仅会产生许多重复代码，而且使代码难于阅读。另外，当需要管理一种新的操作系统时， 你必须在所有的配置清单中进行修改，而不是只修改一处。

更多用法

如果你正在为一个公共的发布（例如 Puppet Forge）编写模块， 使其尽可能的跨平台会让模块变得更有价值。尽你所能，在不同发布、不同平台、不同架构上测试， 并且添加适当的变量，使模块尽可能的应用到各种情况。

如果你正在使用一个公共模块，并修改它适应自己的环境， 如果你认为你的修改可能会帮助到其他人，可以考虑向公共版本提交你的更新。

即使你不想发布你的模块，请铭记：一个模块可能会在生产环境中运行很长一段时间， 并且可能会对其做许多适应环境的改变。 如果从设计模块的一开始就考虑到这些，那么你（或者最终维护你的代码人）的生活将会变得更轻松。

> Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.
>
> — Dave Carhart

参见本书

- 本章的 使用节点继承 一节
- 本章的 使用类的继承和重载 一节
- 第 9 章的 使用公共模块 一节

获得系统的环境信息

> In Paris they simply stared when I spoke to them in French. I never did succeed in making those idiots understand their language.
>
> — Mark Twain

当地的知识是非常有用的。通常在 Puppet 的配置清单中，你需要知道一些所在机器的本地信息。 Puppet 的发行中包含了一个 **Facter** 工具，它提供了从环境中获取系统信息（'facts'） 的一种标准方法，这些信息包括：

- 操作系统
- 内存大小
- 体系结构
- 处理器数量

> note
>
> 要查看关于你的系统中可用的完整的 facts 列表，请运行如下命令：
>
> ```
> # facter
> ```
>
> 虽然它可以方便地从命令行获得信息，但 **Facter** 的真正强大之处在于， 可以在你的 Puppet 配置清单中访问这些 facts。

操作步骤

1. 在你的配置清单中可以像引用其他变量一样引用一个 Facter 的值，例如：

```
notify { "This is $operatingsystem version
$operatingsystemrelease, on $architecture architecture, kernel
version $kernelversion": }
```

2. 一旦运行 Puppet，它就会为当前节点填充适当的值：

```
notice: This is Ubuntu version 10.04, on i386 architecture,
kernel version 2.6.32
```

工作原理

Facter 为 Puppet 提供了一个抽象层，并在配置清单中提供了一种关于环境信息的标准方法。 当你在配置清单中引用一个 fact 值时，Puppet 就会通过查询 Facter 获得当前值， 并将其插入配置清单。

更多用法

你也可以在 **ERB** 模板中使用 facts。例如，你可能会在一个文件中插入一个节点的主机名， 或者基于一个节点的内存大小改变一个应用的配置设置。 当你在模板中使用 fact 的名字时，它们不需要前导的美元符号， 因为在 **ERB** 模板中使用的是 Ruby，而不是 Puppet：

```
$KLogPath <%= case kernelversion when "2.6.31" then "/var/run/rsysl
kmsg" else "/proc/kmsg" end %>
```

参见本书

- 第 9 章的 创建 Facter 的自定义 fact 一节

导入动态信息

尽管有些系统管理员喜欢将他们自己与其它使用成堆的老旧打印机的办公室相隔绝， 但是我们还是需要随时与其他部门交换信息。 例如，你可能需要插入数据到你的 Puppet 配置清单，这些数据是从外部数据源派生出来的。 generate 函数在这方面相当有用。

准备工作

在 Puppetmaster 上使用如下代码创建脚本 /usr/local/bin/latest-puppet.rb：

```
#!/usr/bin/ruby
require 'open-uri'
page = open("http://www.puppetlabs.com/misc/download-options/").rea
print page.match(/stable version is ([\d\.]*)/)[1]
```

操作步骤

1. 在你的配置清单中添加如下的内容：

```
$latestversion = generate("/usr/local/bin/latest-puppet.rb")
notify { "The latest stable Puppet version is ${latestversion}.
You're using ${puppetversion}.": }
```

2. 运行 Puppet：

```
# puppet agent --test
notice: The latest stable Puppet version is 2.6.5\. You're usir
```

工作原理

generate 函数在 Puppetmaster 上（不是客户端）运行指定的脚本或程序并返回结果， 在本例中，返回的结果是 Puppet 最近发布的稳定版的版本号。

我不建议你在生产环境中运行此脚本，因为 Puppet Labs 会随时重新组织其 Web 站点， 这会导致脚本中抓取的 URL 失效，但是你可以从中获得些启发和想法。 一个脚本可以做的一切，比如打印、提取或者计算（例如数据库的查询结果） 都可以使用 generate 将其带入你的配置清单。

值得记住的是：正如在模板中嵌入 Ruby 调用一样，generate 函数运行在 Puppetmaster 上，而不是运行在 Puppet 的客户端节点上。 我曾经犯过一个在模板中调用 /bin/hostname 的错误， 结果让我大吃一惊的是，所有的节点名称都变成了 puppet。

当你需要获得有关节点的特别信息时，最好使用一个自定义的 fact 来实现。

更多用法

如果你在调用 generate 时需要为其传递参数去执行， 可以将这些参数作为函数调用的额外参数，例如：

```
$latestpuppet = generate("/usr/local/bin/latest-version.rb", "puppe
$latestmc = generate("/usr/local/bin/latest-version.rb", "mcollect:
```

你还可以使用 generate 调用 Shell 命令，Puppet 的 generate 使用限制特殊字符的方式来避免恶意的 Shell 调用，例如不能使用 Shell 的管道符（|）。 最简单而安全的办法就是将你所有的逻辑写到一个脚本中，然后使用 generate 调用此脚本。

参见本书

- 第 9 章的 创建 Facter 的自定义 fact 一节
- 本章的 从 CSV 文件导入数据 一节

从 CSV 文件导入数据

想要知道更多东西吗？当你需要从表中查找某些值时，可以使用冗长的 case 语句或 selectors 实现，但更整洁的方式是使用 extlookup 函数实现。 在 puppetmaster 上可以使用 extlookup 函数查询外部的 CSV 文件，并返回匹配的数据片段。

将所有数据组织到一个单一的文件并将它从 Puppet 配置清单中分离出来， 可以使维护工作变得更简单，也便于与其他人分享：例如， 一个开发团队通过部署合适的 CSV 文件作为应用程序发布的一部分， 就可以管理 Puppet 需要知道的有关其应用程序的一切。 Puppet 只需要知道在哪里可以找到这个 CSV 文件，剩下的工作交由 extlookup 去完成。

准备工作

1. 使用如下内容创建 /var/www/apps/common.csv 文件：

```
path,/var/www/apps/%{name}
railsversion,3
domain,www.%{name}.com
```

2. 使用如下内容创建 /var/www/apps/myapp.csv 文件：

```
railsversion,2
```

操作步骤

1. 在你的配置清单中添加如下代码：

```
$extlookup_datadir = "/var/www/apps/"
$extlookup_precedence = [ "%{name}", "common" ]

class app( $name ) {
    $railsversion = extlookup("railsversion")
    $path = extlookup("path")
    $domain = extlookup("domain")
    notify { "App data: Path ${path}, Rails version
  ${railsversion}, domain ${domain}": }
}

class { "app": name =&gt; "myapp" }
```

2. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303129760'
notice: App data: Path /var/www/apps/myapp, Rails version 2,
domain www.myapp.com
notice: /Stage[main]/App/Notify[App data: Path /var/www/apps/
myapp, Rails version 2, domain www.myapp.com]/message: defined
'message' as 'App data: Path /var/www/apps/myapp, Rails versior
domain www.myapp.com'
notice: Finished catalog run in 0.58 seconds
```

工作原理

1. 我们做的第一件事是定义一个变量 $extlookup_datadir，它告诉 extlookup 从哪个目录查找数据文件。你通常会在 site.pp 或者在你设置全局变量的文件中设置这个变量：

```
$extlookup_datadir = "/var/www/apps/"
```

2. 然后，我们告诉 extlookup 在哪里寻找数据文件，以及寻找的优先顺序：

```
$extlookup_precedence = [ "%{name}", "common" ]
```

   这可以是任意长度的数组。当我们生成一个 extlookup 查询时，Puppet 会按顺序扫描每个文件，直到找到请求的值。文件名可以包含变量。 我们期望名为 $name 的变量已经被设置，Puppet 将会使用其值作为要查找的第一个文件名。

3. 下一步，在 app 类中，调用 extlookup 获得一个值：

```
$railsversion = extlookup("railsversion")
```

   现在 extlookup 装置查找一个 CSV 文件并从中读取数据。这个文件保存在 $extlookup_datadir 目录（在本例中是 /var/www/apps），文件名为 %{name}.csv（在本例中是 myapp.csv）。 所以 extlookup 读取文件的完整路径是 /var/www/apps/myapp.csv， 此文件包含了 railsversion,2

   我们已经找到了所需要的值（2），所以 extlookup 将其返回。

4. 下一个 extlookup 调用就不那么幸运了：

```
$path = extlookup("path")
```

extlookup 还是首先查找文件 myapp.csv，但是它找不到匹配 path 变量的值。所以会查找 $extlookup_precedence 列表中的下一个文件, 即文件 common.csv：

```
path,/var/www/apps/%{name}
railsversion,3
domain,www.%{name}.com
```

就本例而言返回值为 /var/www/apps/myapp。

你可以看到，这允许我们在 common.csv 文件中设置一个默认值的集合， 每个应用程序都可以选择其自己的 myapp.csv 文件覆盖 common.csv 文件中的默认值。extlookup 会依次查询列在 $extlookup_precedence 中的文件，直到其找到请求的值。 由于 myapp.csv 被列在首位，所以设置在其中的任何值都优先于文件 common.csv 中的。

更多用法

你也可以在 extlookup 调用中指定默认值，当在所有的 CSV 文件中都没有找到匹配的数据时就会使用这个默认值：

```
$path = extlookup("path", "/var/www/misc")
```

你还可以指定在查找 $extlookup_precedence 所列的文件之前首先要查找的 CSV 文件：

```
$path = extlookup("path", "/var/www/misc", "paths")
```

这首先在 paths.csv 文件中查找数据，如果没找到，将会依次查找列在 $extlookup_precedence 中的其他文件。

CSV 文件中的值也可以引用变量，正如我们曾经看到的：

```
domain,www.%{name}.com
```

你可以使用当前范围内可用的任何变量，包括 Facter 检测出的 facts 值：

```
domain,%{fqdn}
```

R.I. Pienaar 的文章 **“Complex data and Puppet”** 是对 extlookup 的一个很好的介绍： http://www.devco.net/archives/2009/08/31/complex_data_and_puppet.php 。

Jordan Sissel 曾经写过 **“about configuring your whole infrastructure using extlookup”**： http://sysadvent.blogspot.com/2010/12/day-12-scaling-operabilitywith-truth.html 。

参见本书

- 本章的 导入动态信息 一节
- 第 9 章的 创建 Facter 的自定义 fact 一节

给 Shell 命令传递参数

如果你需要在命令行下插入一个值，通常需要使用引号将其括起来，尤其是当其包含空格时更是如此。 shellquote 函数可以携带任意数量的参数（包括数组），它会使用引号将每个参数都括起来并返回一个可以传递到 Shell 命令行上的以空格间隔的字符串。

在下面的例子中，我们想要创建一个用于文件改名的 exec 资源， 然而源文件名和目标文件名都包含空格，因此在命令行上需要正确地使用引号将它们括起来。

操作步骤

1. 在你的配置清单中添加如下的代码：

```
$source = "Hello Jerry"
$target = "Hello... Newman"
$argstring = shellquote( $source, $target )
$command = "/bin/mv ${argstring}"
notify { $command: }
```

2. 运行 Puppet：

```
notice: /bin/mv "Hello Jerry" "Hello... Newman"
```

工作原理

1. 首先我们定义了 $source 和 $target 两个变量， 它们的值是将要在命令行上使用的两个文件名：

```
$source = "Hello Jerry"
$target = "Hello... Newman"
```

2. 然后我们调用 shellquote 串连两个变量为一个以空格间隔的使用引号将每个变量值括起来的字符串：

```
$argstring = shellquote( $source, $target )
```

3. 然后将这其组织到一起，形成最终的命令行：

```
$command = "/bin/mv ${argstring}"
```

结果为：

```
/bin/mv "Hello Jerry" "Hello... Newman"
```

4. 现在可以使用 exec 资源执行这个命令行了。若我们没有使用 shellquote 会发生什么？

```
$source = "Hello Jerry"
$target = "Hello... Newman"
$command = "/bin/mv ${source}  ${target}"
notify { $command: }
```

结果为：

```
notice: /bin/mv Hello Jerry Hello... Newman
```

这将无法工作，因为 mv 命令期望以空格间隔的参数， 所以 mv 将参数解释为请求移动的三个文件：Hello、Jerry 和 Hello… 以及移动的目标目录 Newman，这可不是我们想要的结果。

使用文件和软件包

> If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.
>
> — Gerald Weinberg

在本章中，我们将学习如下内容：

- 为配置文件添加配置行
- 使用 Augeas 自动修改配置文件
- 使用配置片段构建配置文件
- 使用 ERB 模板
- 在模板中遍历数组
- 从第三方仓库安装软件包
- 配置 APT 软件仓库
- 配置 GEM 仓库
- 从源码包自动构建软件
- 比较软件包的版本

作为 Puppet 系统管理员，你所涉及的最多的管理工作就是文件和包。 文件和包是 Puppet 中最重要的两种资源，本章将帮助你深入了解它们， 并学习一些有用的功能和模式，以帮助你更好地使用它们。

在本章中，我们将看到：如何对文件做轻微地编辑； 如何利用 **Augeas** 工具以结构化的方式实现大规模的改变； 如何串联片段构建文件；以及如何从模板生成文件。 我们还将学习，如何从额外的仓库安装包，以及如何创建这些仓库。

为配置文件添加配置行

你知道 Puppet 能做微创手术吗？ 通常，我们不希望将整个配置文件交由 Puppet 管理而仅仅是在配置文件中添加某项设置?—?尤其是如果该文件是由别人管理，我们不能覆盖它的情况。 一种简单而有用的方法是，如果配置文件中不存在指定的行就添加这行配置。例如： 添加一个内核模块名到配置文件 /etc/modules 告知内核在启动时加载此模块。

你可以使用一个 exec 资源达成此任务。下面的例子将展示如何使用 exec 向一个文本文件追加一行配置。

操作步骤

1. 使用如下内容创建 /etc/puppet/manifests/utils.pp 文件：

```
define append_if_no_such_line($file, $line) {
    exec { "/bin/echo '$line' &gt;&gt; '$file'":
        unless =&gt; "/bin/grep -Fx '$line' '$file'",
    }
}
```

2. 添加如下行到 /etc/puppet/manifests/site.pp 文件：

```
import "utils.pp"
```

3. 添加如下代码到你的配置清单：

```
append_if_no_such_line { "enable-ip-conntrack":
    file =&gt; "/etc/modules",
    line =&gt; "ip_conntrack",
}
```

4. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303649606'
notice: /Stage[main]//Node[cookbook]/Append_if_no_such_
line[enable-ip-conntrack]/Exec[/bin/echo 'ip_conntrack' &gt;&gt
modules']/returns: executed successfully
notice: Finished catalog run in 1.22 seconds
```

工作原理

exec 资源将追加指定的文本 $line 到文件 $file，除非它已经存在：

```
exec { "/bin/echo '$line' >> '$file'":
    unless => "/bin/grep -Fx '$line' '$file'",
```

现在，append_if_no_such_line 资源已经可以在你的配置清单中使用了。 在本例中，我们已经使用它确保 /etc/modules 文件（此文件指定在启动过程中需要加载哪些内核模块）包含如下的行：

```
ip_conntrack
```

更多用法

类似的，你可以使用 define 函数对文本文件执行其他细小的操作。例如， 下面的代码片段可以让你在文本文件中执行查找和替换的操作：

```
define replace_matching_line( $match, $replace ) {
    exec { "/usr/bin/ruby -i -p -e 'sub(%r{$match}, \"$replace\")'
 $name":
        onlyif    => "/bin/grep -E '$match' $name",
        logoutput => on_failure,
    }
}

replace_matching_line { "/etc/apache2/apache2.conf":
    match   => "LogLevel .*",
    replace => "LogLevel debug",
}
```

参见本书

- 本章的 使用 Augeas 自动修改配置文件 一节

使用 Augeas 自动修改配置文件

当然，有关标准的事物是如此之多。有时每个应用程序的配置格式都略有不同， 书写正则表达式来解析和修改所有这些配置文件是一项很烦人的工作。

幸好 Augeas 在这方面可以帮助我们。Augeas 是一个旨在简化使用不同配置文件格式工作的工具， 它将不同格式的配置文件统一呈现为一个简单的包含所有配置项的树状结构。 Puppet 的 Augeas 支持允许你创建 augeas 资源，它可以智能地自动地为所需的配置做相应的改变。

准备工作

在使用 Augeas 之前需要先安装它。如下的 Puppet 代码将添加 Augeas 到你的配置。

1. 使用如下内容创建 /etc/puppet/modules/admin/manifests/augeas.pp 文件：

```
class admin::augeas {
    package { [ "augeas-lenses",
                "augeas-tools",
                "libaugeas0",
                "libaugeas-ruby1.8" ]:
        ensure =&gt; "present"
    }
}
```

2. 在一个节点上包含此类：

```
node cookbook {
    include admin::augeas
}
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303657095'
notice: /Stage[main]/Admin::Augeas/Package[augeas-tools]/ensure
ensure changed 'purged' to 'present'
notice: Finished catalog run in 21.96 seconds
```

操作步骤

1. 使用如下内容创建 /etc/puppet/modules/admin/manifests/ipforward.pp 文件：

```
class admin::ipforward {
    augeas { "enable-ip-forwarding":
        context =&gt; "/files/etc/sysctl.conf",
        changes =&gt; [
            "set net.ipv4.ip_forward 1",
        ],
    }
}
```

2. 在一个节点上包含此类：

```
node cookbook {
    include admin::augeas
    include admin::ipforward
}
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303729376'
notice: /Stage[main]/Admin::Ipforward/Augeas[enable-ipforwardir
returns: executed successfully
notice: Finished catalog run in 3.53 seconds
```

4. 使用如下命令检查配置是否生效：

```
# sysctl -p |grep forward
net.ipv4.ip_forward = 1
```

工作原理

下面将描述前面的代码是如何工作的：

1. 我们声明了一个名为 enable-ip-forwarding 的 augeas 资源：

```
augeas { "enable-ip-forwarding":
```

2. 我们指定要实施改变的文件 /etc/sysctl.conf：

```
    context =&gt; "/files/etc/sysctl.conf",
```

3. 将我们需要改变的设置作为一个数组（本例中只有一个数组元素）传递给 changes 参数：

```
    changes =&gt; [
        "set net.ipv4.ip_forward 1",
    ],
```

通常使用 Augeas 改变设置需用如下的格式：

```
set &lt;parameter&gt; &lt;value&gt;
```

Augeas 使用一套名为 **lenses** 的转换文件，使其能够为给定的配置文件以适当的格式改写设置。 在本例中，此设置将转换为 /etc/sysctl.conf 文件中的如下一行：

```
net.ipv4.ip_forward=1
```

更多用法

之所以使用 /etc/sysctl.conf 文件做例子，是因为它包含了各种各样的内核设置， 你可能想要在不同的 Puppet 类中因各种不同的目的而改变这些设置。 在前面的例子中，我们为一个路由器的类启用了 IP 转发， 然而你也可以为一个负载均衡的类调整 net.core.somaxconn 的值。

这意味着简单地对 /etc/sysctl.conf 文件“Puppet化”（"Puppetizing"） 并分发它不能胜任所有的情况，因为你要修改的设置根据不同的服务需求可能会有不同， 从而引发版本冲突。Augeas 则是这种情况的正确解决方案，因为你可以在不同的位置定义 augeas 资源去修改相同的配置文件，这样就不会引起冲突。

Augeas 是个强大的工具，使用 lenses 可以转换绝大多数标准的 Linux 配置文件，而且如果需要管理自己的专有格式的配置文件，你也可以书写自己的专有配置格式。 更多关于使用 Puppet 和 Augeas 的信息，请访问 Puppet Labs 的 wiki 页面：http://projects.puppetlabs.com/projects/1/wiki/Puppet_Augeas 。

使用配置片段构建配置文件

你怎能一口吃头大象呢？有时你会遇到这种情况，想要从被不同类管理的各种片段构建单一的配置文件。 例如，你可能有两到三个服务需要 rsync 模块配置，因此你不能只发布单一的配置文件 rsyncd.conf。 尽管你可以使用 Augeas，但是一种更简单的方法就是使用 exec 资源将若干配置片段串接成一个文件。

操作步骤

1. 使用如下代码创建 /etc/puppet/modules/admin/manifests/rsyncdconf.pp 文件：

```
class admin::rsyncdconf {
    file { "/etc/rsyncd.d":
        ensure =&gt; directory,
    }

    exec { "update-rsyncd.conf":
        command =&gt; "/bin/cat /etc/rsyncd.d/*.conf &gt; /etc/
        refreshonly =&gt; true,
    }
}
```

2. 添加如下代码到你的配置清单：

```
class myapp::rsync {
    include admin::rsyncdconf

    file { "/etc/rsyncd.d/myapp.conf":
        ensure  =&gt; present,
        source  =&gt; "puppet:///modules/myapp/myapp.rsync",
        require =&gt; File["/etc/rsyncd.d"],
        notify  =&gt; Exec["update-rsyncd.conf"],
    }
}
include myapp::rsync
```

3. 使用如下内容创建 /etc/puppet/modules/myapp/files/myapp.rsync 文件：

```
[myapp]
    uid = myappuser
    gid = myappuser
    path = /opt/myapp/shared/data
    comment = Data for myapp
    list = no
    read only = no
    auth users = myappuser
```

4. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303731804'
notice: /Stage[main]/Admin::Rsyncdconf/File[/etc/rsyncd.d]/ensu
created
notice: /Stage[main]/Myapp::Rsync/File[/etc/rsyncd.d/myapp.conf
ensure: defined content as '{md5}e1e57cf38bb88a7b4f2fd6eb1ea282
info: /Stage[main]/Myapp::Rsync/File[/etc/rsyncd.d/myapp.conf]:
Scheduling refresh of Exec[update-rsyncd.conf]
notice: /Stage[main]/Admin::Rsyncdconf/Exec[update-rsyncd.conf]
Triggered 'refresh' from 1 events
notice: Finished catalog run in 1.01 seconds
```

工作原理

admin::rsyncdconf 类为存放 rsync 的配置片段创建了如下目录：

```
file { "/etc/rsyncd.d":
    ensure => directory,
}
```

当你创建一个配置片段（例如本例中的 myapp::rsync），你要做的所有工作就是 require 这个目录（require => File["/etc/rsyncd.d"]）并 notify 资源 exec 更新主配置文件（notify => Exec["update-rsyncd.conf"],）。

只要 /etc/rsyncd.d 目录中有一个配置片段被更新，exec 资源就会执行一次：

```
exec { "update-rsyncd.conf":
    command     => "/bin/cat /etc/rsyncd.d/*.conf > /etc/rsyncd.cor
    refreshonly => true,
}
```

这样，/etc/rsyncd.d 目录中的所有配置片段将会串接成一个 rsyncd.conf 文件。

这是非常有用的，原因在于你可以有许多不同的片段资源散布在各个不同的类或模块中， 所有这些片段最终都将被合并成单一的 rsyncd.conf 文件，而你却可以在一个地方实现配置代码的合并。

更多用法

当你有一个像 rsync 这样的服务且在一个配置文件中需要包含不同的片段时，这会是一种很有用的模式。 实际上，这为你提供了类似于 Apache 的 conf.d 目录或 PHP 的 php-ini.d 目录的功能。

参见本书

- 第 2 章的 Using tags 一节

使用 ERB 模板

模板（**template**）是一种高级别的文本文件。它可以做计算、执行 Ruby 代码或者引用你在 Puppet 配置清单中定义的变量值。能被 Puppet 部署的任何一个文本文件，你都可以使用模板来代替。 最简单的情况，一个模板可以仅是一个静态文本文件。更有用的情况是，你可以使用 **ERB** （嵌入式 **Ruby**）语法在模板中插入变量。例如：

```
<%= name %>, this is a very large drink.
```

如果模板使用了变量 $name，其值为 Zaphod Beeblebrox，模板将解析为：

```
Zaphod Beeblebrox, this is a very large drink.
```

这种简单的技术非常有用，一种情形是要生成一批文件且它们仅有一两个变量的值不同（例如虚拟主机的配置文件）；另一种情形是要向一个脚本中插入值（例如数据库名或口令）。 在下面的例子中，我们将使用一个 ERB 模板插入口令到一个备份脚本。

操作步骤

1. 使用下面的内容创建 /etc/puppet/modules/admin/templates/backup-mysql.sh 文件：

```
#!/bin/sh
/usr/bin/mysqldump -uroot -p&lt;%= mysql_password %&gt; --all-d
| /bin/gzip &gt; /backup/mysql/all-databases.sql.gz
```

2. 添加如下代码到你的配置清单：

```
$mysql_password = "secret"
file { "/usr/local/bin/backup-mysql":
    content =&gt; template("admin/backup-mysql.sh"),
    mode    =&gt; "755",
}
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1308670971'
notice: /Stage[main]//Node[cookbook]/File[/usr/local/bin/backup
ensure: defined content as '{md5}5853b6d4dd72420e341fa7ecb8
91ad43'
notice: Finished catalog run in 0.96 seconds
```

4. 检查 Puppet 是否已经在模板中正确地插入了口令：

```
# cat /usr/local/bin/backup-mysql
```

```
#!/bin/sh
/usr/bin/mysqldump -uroot -psecret --all-databases \
| /bin/gzip &gt; /backup/mysql/all-databases.sql.gz
```

工作原理

在模板中，无论在哪里引用了变量（例如 <%= mysql_password %>），Puppet 都会使用相应的值（例如 secret）替换它。

更多用法

在本例中，我们仅在模板中使用了一个变量，但是只要你需要可以引用许多变量。它们也可以是对 facts 的引用，例如：

```
ServerName <%= fqdn %>
```

或者使用 Ruby 表达式，例如：

```
MAILTO=<%= emails.join(',') %>
```

或者你要使用的任何 Ruby 代码，例如：

```
ServerAdmin <%= sitedomain == 'coldcomfort.com' ? 'seth@coldcomfort
  : 'flora@poste.com' %>
```

参见本书

- 本章的 在模板中遍历数组 一节

在模板中遍历数组

在前面的例子中，我们已经看到使用 Ruby 可以根据表达式的结果插入不同的值。你也可以使用循环对数组中的每个元素生成内容。

操作步骤

1. 添加如下代码到你的配置清单：

```
$ipaddresses = [ '192.168.0.1',
                 '158.43.128.1',
                 '10.0.75.207' ]

file { "/tmp/addresslist.txt":
    content =&gt; template("admin/addresslist.erb")
}
```

2. 使用如下内容创建 /etc/puppet/modules/admin/templates/addresslist.erb 文件：

```
&lt;% ipaddresses.each do |ip| -%&gt;
IP address &lt;%= ip %&gt; is present.
&lt;% end -%&gt;
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304766335'
notice: /Stage[main]//Node[cookbook]/File[/tmp/addresslist.txt]
ensure: defined content as '{md5}7ad1264ebdae101bb5ea0afef474b3
notice: Finished catalog run in 0.64 seconds
```

4. 检查生成文件的内容：

```
# cat /tmp/addresslist.txt
IP address 192.168.0.1 is present.
IP address 158.43.128.1 is present.
IP address 10.0.75.207 is present.
```

工作原理

1. 在模板的第一行，我们引用了一个数组 ipaddresses，并调用了相应的 each 方法：

```
&lt;% ipaddresses.each do |ip| -%&gt;
```

2. 使用 Ruby 创建的这个循环将对数组中的每个元素执行一次。 每循环一次，变量 ip 将会设置成数组当前元素的值。
3. 在我们的例子中，ipaddresses 数组包含三个元素，所以下面的行将执行三次， 对每个数组元素执行一次：

```
IP address &lt;%= ip %&gt; is present.
```

4. 结果将产生如下三行内容：

```
IP address 192.168.0.1 is present.
IP address 158.43.128.1 is present.
IP address 10.0.75.207 is present.
```

5. 最后一行是循环的结束：

```
&lt;% end -%&gt;
```

6. 注意循环的第一行和最后一行都以 -%> 结束，而不是我们以前所看到的以 %> 结束。 使用 - 的作用是要抑制换行，否则会产生换行符，即在文件中产生无用的空白行。

更多用法

模板也可以遍历哈希，或哈希数组，例如：

```
$interfaces = [ { name => 'eth0',
                  ip => '192.168.0.1' },
                { name => 'eth1',
                  ip => '158.43.128.1' },
                { name => 'eth2',
                  ip => '10.0.75.207' } ]

<% interfaces.each do |interface| -%>
Interface <%= interface['name'] %> has the address <%= interface['i
<% end -%>
```

结果为：

```
Interface eth0 has the address 192.168.0.1.
Interface eth1 has the address 158.43.128.1.
Interface eth2 has the address 10.0.75.207.
```

参见本书

- 第 5 章的 使用 ERB 模板 一节

从第三方仓库安装软件包

大多数情况下，你会从发行版本的官方仓库安装软件包，所以使用简单的 package 资源就可以做到：

```
package { "exim4": ensure => installed }
```

但是，有时你需要的软件包只能在第三方仓库（例如 **Ubuntu PPA**）中找到。 或者是第三方仓库提供的软件包版本比官方仓库中的软件包新。

对于手工管理的主机，你通常需要在安装软件包之前先添加仓库源配置到 /etc/apt/sources.list.d （如果有必要，还应该导入仓库的 GPG 公钥）。我们可以很容易地使用 Puppet 自动完成这个过程。

操作步骤

1. 添加如下代码到你的配置清单：

```
package { "python-software-properties": ensure =&gt; installed

exec { "/usr/bin/add-apt-repository ppa:mathiaz/puppet-backport
    creates =&gt; "/etc/apt/sources.list.d/mathiaz-puppet-backp
 list",
    require =&gt; Package["python-software-properties"],
}
```

2. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304773240'
notice: /Stage[main]//Node[cookbook]/Exec[/usr/bin/add-aptrepos
ppa:mathiaz/puppet-backports]/returns: executed
successfully
notice: Finished catalog run in 5.97 seconds
```

工作原理

1. python-software-properties 软件包提供了 add-apt-repository 命令， 使用此命令可以简化添加额外的软件仓库源配置的过程：

```
package { "python-software-properties": ensure =&gt; installed
```

2. 然后我们使用 exec 资源调用这个命令添加要求的配置：

```
exec { "/usr/bin/add-apt-repository ppa:mathiaz/puppet-backport
```

3. 确保 Puppet 不会在每次运行时都执行这个 exec 资源，我们指定了一个由此命令创建的文件， 如果此文件已经存在，则 Puppet 就会忽略 exec 资源的执行：

```
creates =&gt; "/etc/apt/sources.list.d/mathiaz-puppet-backports
```

你可能想要将这个内容与清除 /etc/apt/sources.list.d 中不需要的仓库定义相结合， 正如本书 使用文件资源递归地分发整个目录树 一节中所描述的。

更多用法

这种处理仓库的方法是针对 Debian 和 Ubuntu 系统的，我们之前说过，这是本书使用的参考平台。 如果你使用的是基于 Red Hat 的系统，你可以直接使用 yumrepo 资源管理 RPM 仓库。

参见本书

- 第 6 章的 使用文件资源递归地分发整个目录树 一节

配置 APT 软件仓库

运行自己的软件仓库有几个优点。你可以在自己的仓库中发布自己的软件包。 你可以在自己的软件仓库中放置上游软件包或第三方软件包，从而控制你使用的软件版本。 你可以将自己的软件仓库放置在其他服务器附近，从而避免网速缓慢或镜像站点无法访问的问题。

即使你不需要创建自己的软件包，也可能想要下载特定版本软件包所需的关键依赖包， 并将这些依赖包存储在自己的仓库中，从而防止因上游发生变故而产生的任何意外（例如，你的发行版本已到达生命周期的终结或者上游仓库已经被关闭）。

这也使得通过 Puppet 自动更新软件包便得更容易。你可能偶尔需要更新一个软件包（例如，当有一个安全更新可用时），只要在 package 资源中指定 ensure => latest 就可以方便地实现包更新。但是如果你不能控制仓库，就可能遭遇意想不到的升级风险， 使你的系统受到某种破坏。

使用自己的软件仓库是件两全其美的事情：你可以放心地使用 Puppet 从自己的软件仓库自动更新软件包，当有新版本的软件可用时，只需要将其纳入自己的软件仓库； 你可以首先测试上游的软件版本，确保其可用的情况下再纳入用于生产环境的软件仓库。

准备工作

你需要第 9 章 管理 Apache 服务 一节中讲述的 apache 模块， 如果还没有此模块请先创建它。

在本例中，我将我的仓库命名为 packages.bitfieldconsulting.com。 你可能想要使用一个不同的仓库名，这需要替换本例中的所有的仓库名 packages.bitfieldconsulting.com 为你想要的仓库名。

操作步骤

1. 创建一个新的 repo 模块：

```
# mkdir /etc/puppet/modules/repo
# mkdir /etc/puppet/modules/repo/manifests
# mkdir /etc/puppet/modules/repo/files
```

2. 使用如下内容创建 /etc/puppet/modules/repo/manifests/bitfield-server.pp 文件：

```
class repo::bitfield-server {
    include apache

    package { "reprepro": ensure =&gt; installed }

    file { [ "/var/apt",
             "/var/apt/conf" ]:
        ensure =&gt; directory,
    }

    file { "/var/apt/conf/distributions":
        source  =&gt; "puppet:///modules/repo/distributions",
        require =&gt; File["/var/apt/conf"],
    }

    file { "/etc/apache2/sites-available/apt-repo":
        source  =&gt; "puppet:///modules/repo/apt-repo.conf",
        require =&gt; Package["apache2-mpm-worker"],
    }

    file { "/etc/apache2/sites-enabled/apt-repo":
        ensure  =&gt; symlink,
        target  =&gt; "/etc/apache2/sites-available/apt-repo",
        require =&gt; File["/etc/apache2/sites-available/apt-re
        notify  =&gt; Service["apache2"],
    }
}
```

3. 使用如下内容创建 /etc/puppet/modules/repo/files/distributions 文件：

```
Origin: Bitfield Consulting
Label: bitfield
Suite: stable
Codename: lucid
Architectures: amd64 i386
Components: main non-free contrib
Description: Custom and cached packages for Bitfield Consulting
```

4. 使用如下内容创建 /etc/puppet/modules/repo/files/apt-repo.conf 文件：

```
&lt;VirtualHost *:80&gt;
    DocumentRoot /var/apt
    ServerName packages.bitfieldconsulting.com
    ErrorLog /var/log/apache2/packages.bitfieldconsulting.com.e

    LogLevel warn

    CustomLog /var/log/apache2/packages.bitfieldconsulting.com.

    ServerSignature On

    # Allow directory listings so that people can browse the
    # repository from their browser too
    &lt;Directory "/var/apt"&gt;
        Options Indexes FollowSymLinks MultiViews
        DirectoryIndex index.html
        AllowOverride Options
        Order allow,deny
        allow from all
    &lt;/Directory&gt;

    # Hide the conf/ directory for all repositories
    &lt;Directory "/var/apt/conf"&gt;
        Order allow,deny
        Deny from all
        Satisfy all
    &lt;/Directory&gt;

    # Hide the db/ directory for all repositories
    &lt;Directory "/var/apt/db"&gt;
        Order allow,deny
        Deny from all
        Satisfy all
    &lt;/Directory&gt;
&lt;/VirtualHost&gt;
```

5. 在一个节点的配置清单中添加如下代码：

```
include repo::bitfield-server
```

6. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304775601'

notice: /Stage[main]/Repo::Bitfield-server/File[/var/apt]/ensur
created

notice: /Stage[main]/Repo::Bitfield-server/File[/var/apt/conf]/
ensure: created

notice: /Stage[main]/Repo::Bitfield-server/File[/var/apt/conf/
distributions]/ensure: defined content as '{md5}65dc791b876f533
35fcc42c770283'

notice: /Stage[main]/Repo::Bitfield-server/Package[reprepro]/
ensure: created

notice: /Stage[main]/Repo::Bitfield-server/File[/etc/apache2/
sites-enabled/apt-repo]/ensure: created

notice: /Stage[main]/Repo::Bitfield-server/File[/etc/apache2/
sites-available/apt-repo]/ensure: defined content as '{md5}2da4
957e5acf49220047fe6f6e6e1'

info: /Stage[main]/Repo::Bitfield-server/File[/etc/apache2/site
apt-repo]: Scheduling refresh of Service[apache2]

notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refres
from 1 events

notice: Finished catalog run in 16.32 seconds
```

工作原理

其实，你无需创建一个 APT 仓库。因为可以通过 HTTP 下载软件包，所以你只需要一个 Apache 虚拟主机。 你可以将实际的软件包随意放置在任何地方，只要有一个 conf/distributions 文件并在其中给出 APT 仓库的相关信息。

1. bitfield-server 类的第一部分确保 Apache 已经被设置：

```
class repo::bitfield-server {
    include apache
```

2. reprepro 是用于管理仓库本身的非常有用的工具（例如，添加一个新的软件包）：

```
    package { "reprepro": ensure =&gt; installed }
```

3. 我们创建一个仓库的根目录 /var/apt，以及该目录下的 conf/distributions 文件：

```
    file { [ "/var/apt",
             "/var/apt/conf" ]:
        ensure =&gt; directory,
    }

    file { "/var/apt/conf/distributions":
        source =&gt; "puppet:///modules/repo/distributions",
        require =&gt; File["/var/apt/conf"],
    }
```

4. 这个类的其余部分部署了一个 Apache 虚拟主机的配置文件，用于响应 packages.bitfieldconsulting.com 的请求：

```
file { "/etc/apache2/sites-available/apt-repo":
    source  =&gt; "puppet:///modules/repo/apt-repo.conf",
    require =&gt; Package["apache2-mpm-worker"],
}

file { "/etc/apache2/sites-enabled/apt-repo":
    ensure  =&gt; symlink,
    target  =&gt; "/etc/apache2/sites-available/apt-repo",
    require =&gt; File["/etc/apache2/sites-available/apt-repo"]
    notify  =&gt; Service["apache2"],
}
```

更多用法

当然，一个可用的仓库里不能没有软件包。下面将介绍如何添加软件包， 以及如何配置主机并从你的仓库下载软件包。

向仓库添加软件包

要添加一个软件包到你的仓库，首先下载它然后使用 reprepro 将其添加到仓库：

```
# cd /tmp
# wget http://archive.ubuntu.com/ubuntu/pool/main/n/ntp/\
ntp_4.2.4p8+dfsg-1ubuntu2.1_i386.deb
# cd /var/apt
# reprepro includedeb lucid /tmp/ntp_4.2.4p8+dfsg-1ubuntu2.1_i386.c
Exporting indices...
```

配置节点使用仓库

1. 使用如下内容创建 /etc/puppet/modules/repo/manifests/bitfield.pp 文件（请根据你自己的仓库服务器的 IP 地址替换如下的 IP 地址）：

```
class repo::bitfield {
    host { "packages.bitfieldconsulting.com":
        ip     =&gt; "10.0.2.15",
        ensure =&gt; present,
        target =&gt; "/etc/hosts",
    }

    file { "/etc/apt/sources.list.d/bitfield.list":
        content =&gt; "deb http://packages.bitfieldconsulting.c
        require =&gt; Host["packages.bitfieldconsulting.com"],
        notify  =&gt; Exec["bitfield-update"],
    }

    exec { "bitfield-update":
        command     =&gt; "/usr/bin/apt-get update",
        require     =&gt; File["/etc/apt/sources.list.d/bitfiel
        refreshonly =&gt; true,
    }
}
```

如果你有 DNS 服务器或者你可以控制你的 DNS 区域，可以省略 host 资源的设置。

2. 应用这个类到一个节点：

```
node cookbook {
    include repo::bitfield
}
```

3. 测试你仓库中的 ntp 软件包是否可用：

```
# apt-cache madison ntp
  ntp | 1:4.2.4p8+dfsg-1ubuntu2.1 | http://us.archive.ubuntu.
com/ubuntu/ lucid-updates/main Packages
  ntp | 1:4.2.4p8+dfsg-1ubuntu2.1 | http://packages.
bitfieldconsulting.com/ lucid/main Packages
  ntp | 1:4.2.4p8+dfsg-1ubuntu2 | http://us.archive.ubuntu.
com/ubuntu/ lucid/main Packages
  ntp | 1:4.2.4p8+dfsg-1ubuntu2 | http://us.archive.ubuntu.
com/ubuntu/ lucid/main Sources
  ntp | 1:4.2.4p8+dfsg-1ubuntu2.1 | http://us.archive.ubuntu.
com/ubuntu/ lucid-updates/main Sources
```

对软件包签名

对于生产环境，你应该对软件仓库设置 GPG 密钥并且对软件包进行签名，关于如何设置密钥和签名的信息， 请参考 Sander Marechal 撰写的关于“设置和管理 APT 仓库”的文章：
http://www.jejik.com/articles/2006/09/setting_up_and_managing_an_apt_repository_with_reprepro/ 。

配置 GEM 仓库

使用另外的不兼容的包系统是每个系统管理员的梦想。 如果你要管理 Ruby 或 Rails 应用程序，就需要处理 Rubygems。 与维护自己的 APT 仓库一样，维护你自己的 **gem** 仓库 会有很多优点。 你可以控制包的可用性和包的版本，也可以根据需要发布你自己的 Rubygems。

操作步骤

1. 使用如下内容创建 /etc/puppet/modules/repo/manifests/gem-server.pp 文件：

```
class repo::gem-server {
    include apache

    file { "/etc/apache2/sites-available/gemrepo":
        source  =&gt; "puppet:///modules/repo/gemrepo.conf",
        require =&gt; Package["apache2-mpm-worker"],
        notify  =&gt; Service["apache2"],
    }

    file { "/etc/apache2/sites-enabled/gemrepo":
        ensure  =&gt; symlink,
        target  =&gt; "/etc/apache2/sites-available/gemrepo",
        require =&gt; File["/etc/apache2/sites-available/gemrep
        notify  =&gt; Service["apache2"],
    }

    file { "/var/gemrepo":
        ensure =&gt; directory,
    }
}
```

2. 使用如下内容创建 /etc/puppet/modules/repo/files/gemrepo.conf 文件：

```
&lt;VirtualHost *:80&gt;
    ServerAdmin john@bitfieldconsulting.com
    ServerName gems.bitfieldconsulting.com
    ErrorLog logs/gems.bitfieldconsulting.com-error_log
    CustomLog logs/gems.bitfieldconsulting.com-access_log commo

    Alias / /var/gemrepo/
    &lt;Location /&gt;
        Options Indexes
    &lt;/Location&gt;
&lt;/VirtualHost&gt;
```

3. 在配置清单的节点中添加如下代码：

```
node cookbook {
    include repo::gem-server
}
```

4. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304949279'

notice: /Stage[main]/Repo::Gem-server/File[/etc/apache2/
sites-available/gemrepo]/ensure: defined content as '{md5}
ae1fd948098f14503de02441d02a825d'

info: /Stage[main]/Repo::Gem-server/File[/etc/apache2/sitesavai
gemrepo]: Scheduling refresh of Service[apache2]

notice: /Stage[main]/Repo::Gem-server/File[/etc/apache2/siteser
gemrepo]/ensure: created

info: /Stage[main]/Repo::Gem-server/File[/etc/apache2/sitesenab
gemrepo]: Scheduling refresh of Service[apache2]

notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refres
from 2 events

notice: /Stage[main]/Repo::Gem-server/File[/var/gemrepo]/ensure
created

notice: Finished catalog run in 6.52 seconds
```

工作原理

其原理与 APT 仓库的例子是完全一样的。我们首先定义了一个 gem 仓库的存放目录， 然后定义了一个 Apache 的虚拟主机，使其能够响应 gems.bitfieldconsulting.com 的请求。

更多用法

与 APT 仓库类似，仅有空仓库是无用的，下面将演示如何将 gem 包纳入仓库， 以及如何配置你的节点访问自己的 gem 仓库。

添加 gem 仓库

向你的仓库添加新的 gems 包很简单。将 gem 文件存入 /var/gemrepo/gems 目录， 然后在 /var/gemrepo 目录下运行如下命令即可：

```
# gem generate_index
```

使用 **gem** 仓库

像使用 APT 仓库一样，首先确保你的节点能够解析主机名 gems.bitfieldconsulting.com， 既可以使用 Puppet 部署 host 资源，也可以通过配置 DNS 实现。

然后在 Puppet 中使用如下代码指定一个 gem 软件包：

```
package { "json":
    provider => "gem",
    source => "http://gems.bitfieldconsulting.com ",
}
```

从源码包自动构建软件

源码压缩包（**tarball**）会严重损害你的健康。你既可以使用 发行版（**distro**）或第三方软件包， 同时从源码包构建你自己的软件包也是可取的，这有时还会有许多工作要做。 创建 Debian 软件包（或者其他风格的软件包）会是一个漫长而容易出错的过程， 而且你可能没有时间或预算构建它。

如果你必须从源代码构建你的程序，至少 Puppet 可以帮你实现这一过程。 一般地构建过程是自动化的，否则你就得手工构建：

- 下载源码包
- 解压源码包
- 配置（Configure）并构建（build）程序
- 安装已构建的程序

在下面的例子中，我们将从源码构建 **OpenSSL**（虽然在生产环境中你应该使用发行版提供的软件包， 但这是一个有用的演示示例）。

操作步骤

1. 添加如下代码到你的配置清单：

```
exec { "build-openssl":
    cwd =&gt; "/root",
    command =&gt; "/usr/bin/wget ftp://ftp.openssl.org/source/
 openssl-0.9.8p.tar.gz && /bin/tar xvzf openssl-0.9.8p.tar.
 gz && cd openssl-0.9.8p && ./Configure linux-generic32 &&
 make install",
    creates =&gt; "/usr/local/ssl/bin/openssl",
    logoutput =&gt; on_failure,
    timeout =&gt; 0,
}
```

2. 运行 Puppet （这可能会执行一段时间！）：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304954159'

notice: /Stage[main]//Node[cookbook]/Exec[build-openssl]/returr
executed successfully

notice: Finished catalog run in 554.00 seconds
```

工作原理

exec 命令被 && 运算符分割成了五个阶段。也就是说每个阶段的命令若执行失败，整个命令将会失败且停止运行。 当你想确保每个子命令都成功运行之后才运行下一个子命令时，这是一种有用的构造方法。

1. 第一阶段，下载源码包：

   ```
   /usr/bin/wget ftp://ftp.openssl.org/source/openssl-0.9.8p.tar.g
   ```

2. 第二阶段，解压源码包：

   ```
   /bin/tar xvzf openssl-0.9.8p.tar.gz
   ```

3. 第三阶段，进入源码树的目录：

   ```
   cd openssl-0.9.8p
   ```

4. 第四阶段，运行配置脚本（通常还会指定一些自定义选项）：

   ```
   ./Configure linux-generic32
   ```

5. 最后阶段，创建并安装软件：

   ```
   make install
   ```

6. 由于这是一个漫长的过程，你不想每次运行 Puppet 时都运行它，我们指定了一个条件， 即被创建的文件是否应经存在：

```
creates =&gt; "/usr/local/ssl/bin/openssl",
```

无论出于任何原因，如果需要重建，你可以删除此文件。

7. 错误并非只发生在首次编译的情况。在出现问题时，我们指定了 logoutput 参数， 它将为我们显示编译或构建过程中的出错信息：

```
    logoutput =&gt; on_failure,
```

8. 最后，因为编译过程可能会持续一段时间，所以将 timeout 参数设置成了 0（默认情况下，Puppet 执行 exec 命令 5 分钟后就会超时）：

```
    timeout =&gt; 0,
```

更多用法

如果你有一批必须从源代码构建的软件，那么将上述的 exec 放入一个 define 函数会使这一构建过程变得更方便，因为你可以使用大致相同的代码构建每个包。

比较软件包的版本

软件包的版本号是个奇怪的东西。它们看起来像十进制的数字，但它们不是。 例如，一个版本号通常的形式为 2.6.4。如果你需要比较两个版本号， 不能做简单的字符串比较，因为 2.6.4 会比 2.6.12 大；也不能进行数字比较， 因为它们不是有效地数字。

Puppet 的 versioncmp 函数会帮我们解决这个问题。 若你给它传递两个版本号，它会比较它们，并返回一个值，指出谁是更大的：

```
versioncmp( A, B )
```

将返回如下值：

- 若 A 和 B 相同，则返回 0
- 若 A 大于 B ，则返回大于 1 的值
- 若 A 小于 B ，则返回小于 0 的值

操作步骤

1. 添加如下代码到你的配置清单：

```
$app_version = "1.2.2"
$min_version = "1.2.10"

if versioncmp( $app_version, $min_version ) &gt;= 0 {
    notify { "Version OK": }
} else {
    notify { "Upgrade needed": }
}
```

2. 运行 Puppet：

```
notice: Upgrade needed
```

3. 现在更变 $app_version 的值：

```
$app_version = "1.2.14"
```

4. 在次运行 Puppet：

```
notice: Version OK
```

工作原理

我们指定了可接受的最小版本（$min_version）是 1.2.10。 所以在上面的例子中，我们用 $app_version 为 1.2.2 的版本号与之比较。

简单的对两个字符串进行比较（例如，Ruby 中的字符串比较）将会给出错误的结果， 然而 versioncmp 却可以正确地检测出 1.2.2 小于 1.2.10 并提示我们 Upgrade needed。

当改变 $app_version 的值为 1.2.14 后，由于 versioncmp 正确地检测出 $app_version 大于 $min_version，所以提示我们 Version OK。

用户和虚拟资源

> How good the design is doesn't matter near as much as whether the design is getting better or worse. If it is getting better, day by day, I can live with it forever. If it is getting worse, I will die.
>
> — Kent Beck

在本章中，我们将学习如下内容：

- 使用虚拟资源
- 使用虚拟资源管理用户
- 管理用户基于密钥的 SSH 访问
- 管理用户的自定义文件
- 有效地分发 cron 任务
- 当文件更新时运行命令
- 使用主机资源
- 为文件资源指定多个源
- 使用文件资源递归地分发整个目录树
- 清理过期的旧文件
- 使用日程表资源
- 资源的审计
- 临时禁用资源
- 管理时区

维护用户是件痛苦的事，我不是说人，但毫无疑问某种情况下这确实是真的。 为了在网络中的机器上同步 UNIX 用户账号和权限文件，其中一些机器可能运行了不同的操作系统， 没有某种集中的配置管理是非常具有挑战性的。

考虑一个新的开发者要加入开发团队的情况，他在每台机器上都需要一个账号， 并为此账号的组成员分配 sudo 特权；还需要他的 SSH 密钥授权一堆不同的账号。 系统管理员若手工实现这些工作，恐怕要忙活一整天。 使用 Puppet 的系统管理员却能在几分钟内完成这些工作，提早就去吃午餐了。

在本章中，我们将会看到管理用户及其相关资源的处理模式和技术。 我们还将看到：如何在 Puppet 中安排资源的执行； 为提高效率如何将 cron 任务分散到不同的时间执行； 如何处理时区，如何处理 /etc/hosts 条目； 以及如何让 Puppet 收集审计数据，使有人搞乱了你网络中的机器时能被发现。

使用虚拟资源

什么是虚拟资源（Virtual Resource），我们为什么需要它们？ 下面我们来看一个可能会使用虚拟资源的典型例子。 你负责管理 facesquare 和 twitstagram 两个应用程序，他们都是运行在 Apache 上的 Web 应用程序。 facesquare 的定义看起来可能像这样：

```
class app::facesquare
{
    package { "apache2-mpm-worker": ensure => installed }
}
```

twitstagram 的定义看起来可能像这样：

```
class app::twitstagram
{
    package { "apache2-mpm-worker": ensure => installed }
}
```

一切都很好，直到你需要将两个应用程序同时应用到单台服务器上：

```
node micawber
{
    include app::facesquare
    include app::twitstagram
}
```

现在 Puppet 会出错，因为你试图用相同的名字 apache2-mpm-worker 定义两个 package 资源。 错误输出信息如下：

```
err: Could not retrieve catalog from remote server: Error 400 on SE
Duplicate definition: Package[apache2-mpm-worker] is already define
file /etc/puppet/modules/app/manifests/facesquare.pp at line 2; car
redefine at /etc/puppet/modules/app/manifests/twitstagram.pp:2 on r
cookbook.bitfieldconsulting.com
```

你可以从其中的一个类中移除重复的包定义，但是这样话，如果试图在另一个服务器包含 app 类时，就会因为没有准备好 Apache 而失败。

通过在自己的类中放置 Apache 的包资源并使用 include apache 包含它，你就可以解决这个问题， 因为 Puppet 不介意多次包含一个相同的类。但是这有一个缺点，即每个具有潜在冲突的资源都必须有它自己的类。 虚拟资源可以解决这个问题。虚拟资源就像是个普通的资源，特别之处在于它以 @ 字符开始，例如：

```
@package { "apache2-mpm-worker": ensure => installed }
```

你可以把它看作是个“FYI（仅供参考）”资源：我只是告诉 Puppet 这个资源存在，但不希望用它做任何事情。 Puppet 将会读取并记住虚拟资源定义，但实际上不会创建这个资源，直到你明确指出要创建此资源。

要创建这个资源，使用如下的 realize 函数：

```
realize( Package["apache2-mpm-worker"] )
```

对于你想要的资源，可以多次调用 realize 而且不会产生冲突。 因此，虚拟资源用于：当在几个不同的类中都需要相同的资源，且它们可能会在相同的节点上共存的情况。

操作步骤

1. 创建名为 app 的新模块：

   ```
   # mkdir -p /etc/puppet/modules/app/manifests
   ```

2. 使用如下内容创建 /etc/puppet/modules/app/manifests/facesquare.pp 文件：

   ```
   class app::facesquare
   {
       realize( Package["apache2-mpm-worker"] )
   }
   ```

3. 使用如下内容创建 /etc/puppet/modules/app/manifests/twitstagram.pp 文件：

   ```
   class app::twitstagram
   {
       realize( Package["apache2-mpm-worker"] )
   }
   ```

4. 使用如下内容创建 /etc/puppet/modules/admin/manifests/virtualpackages.pp 文件：

   ```
   class admin::virtual-packages
   {
       @package { "apache2-mpm-worker": ensure =&gt; installed }
   }
   ```

5. 在一个节点上包含如下代码：

```
node cookbook
{
    include admin::virtual-packages
    include app::facesquare
    include app::twitstagram
}
```

6. 运行 Puppet。

工作原理

你可以在 admin::virtual-packages 类中定义一个包的虚拟资源。 所有节点都可以包含这个类，并且你可以将所有虚拟资源都放在此类中。 这些虚拟资源都不会实际安装到节点上，直到你调用 realize：

```
class admin::virtual-packages
{
    @package { "apache2-mpm-worker": ensure => installed }
}
```

每个需要 Apache 包的类都可以对虚拟资源调用 realize ：

```
class app::twitstagram
{
    realize( Package["apache2-mpm-worker"] )
}
```

Puppet 知道如何处理它，因为你设置了相应的虚拟资源，你打算多次引用同一个包, 而不会意外地创建具有相同名子的两个资源。所以，这正确地实现了我们的需求。

更多用法

为了实现（realize）虚拟资源，你也可以使用 **collection** 语法：

```
Package <| title = "apache2-mpm-worker" |>
```

使用这种语法的好处是，你不仅可以指定资源名，而且可以指定 tag，例如：

```
Package <| tag = "security" |>
```

或者，你可以指定资源类型的所有实例，在查询部分保留一个空格即可：

```
Package <| |>
```

参见本书

- 本章的 使用虚拟资源管理用户 一节

使用虚拟资源管理用户

用户管理是使虚拟资源能派上用场的另一个很好的例子。考虑下面的设置。 你有三个用户：John、Graham 和 Steven。为了简化管理大量主机，你为两种类型的用户定义了类： developers 和 sysadmins。每个机器都需要包含 sysadmins，但只有一部分机器允许 developer 访问：

```
node server
{
    include user::sysadmins
}

node webserver inherits server
{
    include user::developers
}
```

John 是个系统管理员，Steven 是个开发者，而 Graham 既是系统管理员又是开发者， 所以 Graham 必须是两个组中的成员 。这将在 webserver 上产生冲突，因为最终会导致重复定义用户 Graham。

为了避免这种情况，常见的做法是将所有用户设置成虚拟资源，并定义在一个单独的类 user::virtual 中，每个机器都包含这个类，然后在需要时对虚拟用户执行 realize。

操作步骤

1. 创建如下的用户模块：

   ```
   # mkdir -p /etc/puppet/modules/user/manifests
   ```

2. 使用如下内容创建 /etc/puppet/modules/user/manifests/virtual.pp 文件：

   ```
   class user::virtual
   {
       @user { "john": }
       @user { "graham": }
       @user { "steven": }
   }
   ```

3. 使用如下内容创建 /etc/puppet/modules/user/manifests/developers.pp 文件：

```
class user::developers
{
    realize( User["graham"],
             User["steven"] )
}
```

4. 使用如下内容创建 /etc/puppet/modules/user/manifests/sysadmins.pp 文件：

```
class user::sysadmins
{
    realize( User["john"],
             User["graham"] )
}
```

5. 在一个节点中添加如下代码：

```
include user::virtual
include user::sysadmins
include user::developers
```

6. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305554239'
notice: /Stage[main]/User::Virtual/User[john]/ensure: created
notice: /Stage[main]/User::Virtual/User[steven]/ensure: created
notice: /Stage[main]/User::Virtual/User[graham]/ensure: created
notice: Finished catalog run in 2.36 seconds
```

工作原理

每个节点都应该包含 user::virtual 类，作为基本配置的一部分，所有服务器都继承这个类。这个类将为你的组织或站点定义所有的用户。同时也应包括只是为了运行应用程序或服务的任何用户（即非登录用户，例如 apache 或 git）。

然后你可以对你的用户进行分组（不同于 UNIX 系统中的组，只是用于区分不同的团队或任务角色），例如 developers 和 sysadmins。对于一个组的类，可以 realize 组里需要的所有用户，例如：

```
class user::sysadmins
{
    realize( User["john"],
             User["graham"] )
}
```

之后便可以在任何需要的地方包含这些组，而不用担心因为多次定义同一个用户而产生的冲突。

参见本书

- 本章的 使用虚拟资源 一节
- 本章的 管理用户的自定义文件 一节

管理用户基于密钥的 SSH 访问

唯一安全的服务器是关闭的。尽管如此，对于服务器访问控制的一个好方法是，使用经过私钥短语保护的 SSH 密钥的用户帐户， 而不是多个用户使用一个共享账号和一个众所周知的口令。 Puppet 使这种管理变得简单，感谢其内置的 ssh_authorized_key 类型。

将它与上一节讲述的虚拟用户相结合，你可以创建一个包括 user 和 ssh_authorized_key 的 define。 对于需要添加自定义文件和其他每用户自己的资源时，这也会是有用的。

操作步骤

1. 对上一节创建的 user::virtual 类做如下修改：

```
class user::virtual
{
    define ssh_user( $key )
    {
        user { $name:
            ensure =&gt; present,
            managehome =&gt; true,
             }

        ssh_authorized_key { "${name}_key":
            key =&gt; $key,
            type =&gt; "ssh-rsa",
            user =&gt; $name,
                              }
    }

    @ssh_user { "phil":
        key =&gt; "AAAAB3NzaC1yc2EAAAABIwAAAIEA3ATqENg+GW
 ACa2BzeqTdGnJhNoBer8x6pfWkzNzeM8Zx7/2Tf2pl7kHdbsiT
 XEUawqzXZQtZzt/j3Oya+PZjcRpWNRzprSmd2UxEEPTqDw9LqY5S2B8og/
 NyzWaIYPsKoatcgC7VgYHplcTbzEhGu8BsoEVBGYu3IRy5RkAcZik=",
    }
}
```

2. 在一个节点上包含如下代码：

```
realize( User::Virtual::Ssh_user["phil"] )
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305561740'

notice: /Stage[main]/User::Virtual/User::Virtual::Ssh_user[phil
User[phil]/ensure: created

notice: /Stage[main]/User::Virtual/User::Virtual::Ssh_user[phil
Ssh_authorized_key[phil_key]/ensure: created

notice: Finished catalog run in 1.04 seconds
```

工作原理

我们创建了一个名为 ssh_user 的 define，它包括用户自身的 user 资源以及与其相关的 ssh_authorized_key 资源，内容如下：

```
define ssh_user( $key )
{
    user { $name:
        ensure => present,
        managehome => true,
         }

    ssh_authorized_key { "${name}_key":
        key  => $key,
        type => "ssh-rsa",
        user => $name,
                        }
}
```

然后我们对用户 phil 创建了 ssh_user 的一个虚拟资源实例：

```
    @ssh_user { "phil":
        key => "AAAAB3NzaC1yc2EAAAABIwAAAIEA3ATqENg+GW
 ACa2BzeqTdGnJhNoBer8x6pfWkzNzeM8Zx7/2Tf2pl7kHdbsiT
 XEUawqzXZQtZzt/j3Oya+PZjcRpWNRzprSmd2UxEEPTqDw9LqY5S2B8og/
 NyzWaIYPsKoatcgC7VgYHplcTbzEhGu8BsoEVBGYu3IRy5RkAcZik=",
    }
```

回顾一下，因为资源是虚拟的，Puppet 会意识到这一点，所以它不会创建任何实际的资源， 直到你调用 realize 时才真正创建。

最后，我们在节点中添加了如下代码：

```
    realize( User::Virtual::Ssh_user["phil"] )
```

这会实际创建 user 资源以及包含其公钥的 authorized_keys 资源。

更多用法

可以将这种思想应用到上一节提及的“组织用户到组的类”中，修改相应类的代码为：

```
class user::sysadmins
{
    search User::Virtual

    realize( Ssh_user["john"],
             Ssh_user["graham"] )
}
```

使用 search User::Virtual 的目的仅仅是为了避免杂乱，这允许你直接引用 Ssh_user， 而不用每次都要使用 User::Virtual:: 前缀。

另外，你可能会遭遇如下的错误：

```
err: /Stage[main]/User::Virtual/User::Virtual::Ssh_user[graham]/Ssh
authorized_key[graham_key]: Could not evaluate: No such file or di
- /home/graham/.ssh
```

这或许是因为你之前已经创建过 graham 用户，但没有使用 Puppet 管理其自家目录。 这种情况下，Puppet 不会自动创建 authorized_keys 文件所需的 .ssh 目录。运行如下的命令：

```
# userdel graham
```

之后再次运行 Puppet 即可解决此问题。

管理用户的自定义文件

用户，与猫一样，常常觉得有必要标记他们的领地。 与猫不同，用户往往要定制自己的 shell 环境，如终端显示的颜色、别名等。 这通常是通过用户家目录下的一些以点开始的环境文件实现的，例如：.bash_profile。

通过修改上一节的 user::virtual::ssh_user 类，你可以将环境文件添加到基于 Puppet 的用户管理中。 在这个类中，你可以有选择地为客户提供 Puppet 文件仓库中的任何以点开头的文件。

操作步骤

1. 对 user::virtual 类做如下修改：

```
class user::virtual
{
  define user_dotfile( $username )
  {
    file { "/home/${username}/.${name}":
      source =&gt; "puppet:///modules/user/${username}-${name}"
      owner  =&gt; $username,
      group  =&gt; $username,
    }
  }

  define ssh_user( $key, $dotfile = false )
  {
    user { $name:
      ensure     =&gt; present,
      managehome =&gt; true,
    }

    ssh_authorized_key { "${name}_key":
      key  =&gt; $key,
      type =&gt; "ssh-rsa",
      user =&gt; $name,
    }

    if $dotfile {
      user_dotfile { $dotfile:
          username =&gt; $name,
      }
    }
  }

  @ssh_user { "john":
    key =&gt; "AAAAB3NzaC1yc2EAAAABIwAAAIEA3ATqENg
 +GWACa2BzeqTdGnJhNoBer8x6pfWkzNzeM8Zx7/2Tf2pl7kHdbsi
 TXEUawqzXZQtZzt/j3Oya+PZjcRpWNRzprSmd2UxEEPTqDw9LqY5S2B8
 og/NyzWaIYPsKoatcgC7VgYHplcTbzEhGu8BsoEVBGYu3IRy5RkAcZik=",
    dotfile =&gt; [ "bashrc", "bash_profile" ],
  }
}
```

2. 使用如下内容创建 /etc/puppet/modules/user/files/john-bashrc 文件：

```
export PATH=$PATH:/var/lib/gems/1.8/bin
```

3. 使用如下内容创建 /etc/puppet/modules/user/files/john-bash_profile 文件：

```
. ~/.bashrc
```

4. 运行 Puppet。

工作原理

我们添加了一个名为 user_dotfile 的 define。 对于用户要使用的每个 dotfile 都要调用这个 define 一次。 在本例中，john 用户有两个 dotfiles：.bashrc 和 .bash_profile。 其声明如下：

```
@ssh_user { "john":
    key => ...
    dotfile => [ "bashrc", "bash_profile" ],
}
```

你既可以指定一个 dotfile，也可以像本例那样指定一个数组形式的列表。

对于每个 dotfile，user_dotfile 将在 modules/user/files 目录下查找相应源的文件。 例如，对名为 bashrc 的 dotfile，Puppet 将查找：

```
modules/user/files/john-bashrc
```

这将被拷贝为客户节点的如下文件：

```
/home/john/.bashrc
```

参见本书

- 本章的 使用虚拟资源管理用户 一节

有效地分发 cron 任务

当你有许多服务器需要执行相同的 cron 作业时，不在同一时间运行它们通常是个好主意。 如果所有作业都要访问一个公共服务器，就会给该服务器带来大量负载，即使这些服务器不会同时访问公共服务器，所有服务器也会在同一时间处于繁忙状态， 这可能会削减它们提供其他服务的能力。

Puppet 的 inline_template 函数允许我们使用 Ruby 的逻辑根据主机名为 cron 作业设置不同的运行时间。

操作步骤

1. 在一个节点中添加如下代码：

```
define cron_random( $command, $hour )
{
  cron { $name:
    command =&gt; $command,
    minute  =&gt; inline_template("&lt;%= (hostname+name).hash.
    hour    =&gt; $hour,
    ensure  =&gt; "present",
  }
}

cron_random { "hello-world":
    command =&gt; "/bin/echo 'Hello world'",
    hour =&gt; 2,
}

cron_random { "hello-world-2":
    command =&gt; "/bin/echo 'Hello world'",
    hour =&gt; 1,
}
```

2. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305713506'
notice: /Stage[main]//Node[cookbook]/Cron_random[hello-world]/
Cron[hello-world]/ensure: created
notice: /Stage[main]//Node[cookbook]/Cron_random[hello-world-2]
Cron[hello-world-2]/ensure: created
notice: Finished catalog run in 1.07 seconds
```

3. 检查 crontab 查看是否成功地配置了 cron 作业：

```
# crontab -l
# HEADER: This file was autogenerated at Fri Jul 29 10:58:45 +0
2011 by puppet.
# HEADER: While it can still be managed manually, it is definit
not recommended.
# HEADER: Note particularly that the comments starting with
'Puppet Name' should
# HEADER: not be deleted, as doing so could cause duplicate cro
jobs.
# Puppet Name: hello-world
25 2 * * * /bin/echo 'Hello world'
# Puppet Name: hello-world-2
49 1 * * * /bin/echo 'Hello world'
```

工作原理

我们想要为每个 cron 作业选择一个 随机的 执行分钟数； 而不是真正的随机 （或者说，不是每次运行 Puppet 都会改变 cron 作业的运行时间）， 但这也或多或少地保证了每个主机上的每个 cron 作业运行时间的不同。

我们可以使用 Ruby 的 hash 方法实现它，它会对任何对象（本例为一个字符串）计算出一个哈希值。 尽管看上去这个哈希值是随机的，但它每次运行时都相同，所以当再次运行 Puppet 时其值不会改变。

哈希值生成的是一个大整数，而我们想要的是一个 0 到 59 之间的整数，所以我们使用了 Ruby 的 % （模）运算符将其结果限制在这个范围内。因为只有 60 种可能的值，尽管 hash 函数被设计为尽可能产生随机的输出，还是会有些许的碰撞而且这些碰撞对于 minute 应该是均匀分布的。

因为我们希望每个哈希值在不同的主机上是不同的，所以使用主机名做 hash 处理。 然而，我们还希望同一台主机上的不同作业的哈希值也不同，所以联合使用了主机名和作业名 （例如 hello-world）做 hash 处理。

更多用法

在本例中，我们仅对 cron 作业的 minute 进行了随机化，并将 hour 作为 define 定义的一部分。若你同时希望指定要在周几运行，可以在 cron_random 中添加一个附加参数来指定， 可以像下面这样为其指定默认值：

```
define cron_random( $command, $hour, $weekday = "*" ) {
```

若你想要对 cron 作业的 hour 进行随机化（例如：要做的作业可以在一天之内的任何时间执行， 并且必须将它们均匀分布在所有的 24 个小时上），可以对 cron_random 做如下修改：

```
hour => inline_template("<%= (hostname+name).hash.abs % 24 %>"),
```

参见本书

- 第 1 章的 从 cron 运行 Puppet 一节

当文件更新时运行命令

当某个特定的文件更新后 Puppet 就该采取一些行动，这是一个非常常见的模式。例如，在 rsync 配置片段的例子中，一旦修改了某个片段文件，就会调用 exec 资源更新主配置文件 rsyncd.conf。

每次运行 Puppet，exec 资源都会被运行，除非指定了如下参数中的一个：

- creates
- onlyif
- unless
- refreshonly => true

refreshonly 参数的意思是：仅当从其他资源（例如一个文件资源）获得一个 notify 才执行 exec 资源。

准备工作

安装 nginx 包（实际上，我们只需要配置文件，但这是获得它的最简单方式）:

```
# apt-get install nginx
```

操作步骤

1. 创建一个 nginx 模块，其目录结构如下：

   ```
   # mkdir /etc/puppet/modules/nginx
   # mkdir /etc/puppet/modules/nginx/files
   # mkdir /etc/puppet/modules/nginx/manifests
   ```

2. 使用如下内容创建 /etc/puppet/modules/nginx/manifests/nginx.pp 文件：

```
class nginx {
    package { "nginx": ensure =&gt; installed }

    service { "nginx":
        enable =&gt; true,
        ensure =&gt; running,
    }

    exec { "reload nginx":
        command     =&gt; "/usr/sbin/service nginx reload",
        require     =&gt; Package["nginx"],
        refreshonly =&gt; true,
    }

    file { "/etc/nginx/nginx.conf":
        source  =&gt; "puppet:///modules/nginx/nginx.conf",
        notify  =&gt; Exec["reload nginx"],
        require =&gt; Package["nginx"],
    }
}
```

3. 复制系统中的 nginx.conf 文件到模块的相应目录：

```
cp /etc/nginx/nginx.conf /etc/puppet/modules/nginx/files
```

4. 添加如下代码到你的配置清单：

```
include nginx
```

5. 对复制到 Puppet 中的 nginx.conf 文件做个小改动，以便进行后续的测试：

```
# echo \# &gt;&gt;/etc/puppet/modules/nginx/files/nginx.conf
```

6. 运行 Puppet：

```
# puppet agent --test

info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1303745502'

--- /etc/nginx/nginx.conf 2010-02-15 00:16:47.000000000 -0700
+++ /tmp/puppet-file20110425-31239-158xcst-0 2011-04-25
09:39:49.586322042 -0600
@@ -48,3 +48,4 @@
 #         proxy       on;
 #     }
 # }
+#

info: FileBucket adding /etc/nginx/nginx.conf as {md5}7bf139588
cd5956f986c9c1442d44

info: /Stage[main]/Nginx/File[/etc/nginx/nginx.conf]:
Filebucketed /etc/nginx/nginx.conf to puppet with sum
7bf139588b5ecd5956f986c9c1442d44

notice: /Stage[main]/Nginx/File[/etc/nginx/nginx.conf]/content:
content changed '{md5}7bf139588b5ecd5956f986c9c1442d44' to '{md
d28d08925174c3f6917a78797c4cd3cc'

info: /Stage[main]/Nginx/File[/etc/nginx/nginx.conf]: Schedulin
refresh of Exec[reload nginx]

notice: /Stage[main]/Nginx/Exec[reload nginx]: Triggered 'refre
from 1 events

notice: Finished catalog run in 1.69 seconds
```

工作原理

对于大多数服务来说，你应该简单地定义一个服务资源，它从配置文件获得 notify。 这会使 Puppet 重启服务，从而应用改变之后的更新。

然而，nginx 有时不能正确的重新启动，尤其是当使用 Puppet 重新启动时， 所以我为某个站点炮制了一个让 Puppet 运行 /etc/init.d/nginx reload 的补救措施， 用它替代服务的重启。下面阐述它是如何工作的。

将 exec 资源中的 refreshonly 参数设置成 true：

```
exec { "reload nginx":
    command     => "/usr/sbin/service nginx reload",
    require     => Package["nginx"],
    refreshonly => true,
}
```

所以，仅当它获得通知才会运行。

如果配置文件发生改变，就提供所需的通知（notify）：

```
file { "/etc/nginx/nginx.conf":
    source => "puppet:///modules/nginx/nginx.conf",
    notify => Exec["reload nginx"],
}
```

每当 Puppet 更新了这个配置文件，它就会运行 exec， 这将会调用如下命令重新加载修改后的配置文件：

```
/usr/sbin/service nginx reload
```

如果服务支持 reload 命令参数，就会在不中断服务的情况下为 daemon 发出一个重新读取配置文件的进程信号。

实际上，对于本例，更好的方法是为 nginx 服务定义如下的 restart 命令：

```
service { "nginx":
    restart => "/etc/init.d/nginx reload",
}
```

但是我想要和你分享一些我写的用于展示 notify -> Exec 技术的实际代码， 并且那时，我还不知道 restart 而且它也不存在。 不过作为一种通用的模式，当更新文件后需要采取某种行动时，你会发现它有用。

更多用法

每当遇到资源更新就要采取某些行动的情况，你就可以使用这个类似的模式。可能的用途包括：

- 触发服务重新加载配置文件
- 运行语法检查，然后再重新启动服务
- 连接 config 片段
- 运行测试

- 链接 exec 资源

如果当一个文件（假设名为 somefile）更新后需要执行多个命令，那么在每个命令的 exec 资源声明中都使用 subscribe => File[somfile], （即一旦检测到 somefile 文件的变化就重新执行 exec 资源）会更简单， 而不是在 file 资源中使用 notify 执行命令。 效果是一样的。

> 译者注
>
> 你也可以在 service 资源定义中使用 subscribe， 例如在一个 Redhat 风格的系统中，你可以使用如下的 service 声明：
>
> ```
>     service { "nginx":
>         enable    => true,
>         ensure    => running,
>         restart   => "/etc/init.d/nginx reload",
>         subscribe => [ File["/etc/nginx/nginx.conf"],
>                        File["/etc/sysconfig/nginx"] ]
>     }
> ```
>
> 在上例中，使用数组为 subscribe 指定了两个要检测的文件， 也就是说，一旦检测到两个文件中的任何一个发生变化就重启服务。

使用主机资源

> I am not a number.
>
> *"The Prisoner"* — Number Six

将机器搬来搬去是很常见的做法，尤其是在云的基础设施中搬动，所以一个特定主机的 IP 经常会改变。 正因为如此，在你的配置中使用硬编码的 IP 地址显然是个坏主意。 如果一台机器要访问另一台（例如，一个应用服务器需要访问一台数据库服务器），那么使用主机名而不是 IP 地址会更好。

然而，如何映射主机名到 IP 地址呢？这通常会使用 DNS，但小型组织里一般没有 DNS 服务器， 而大型组织要想无困扰地实现 DNS 更新也是相当费时的。 此外，将 DNS 信息传播到不同机器也需要不同的时间，因此确保快速而一致的 IP 地址更新的一种方法是使用由 Puppet 管理的本地主机表文件 /etc/hosts 中的条目。

操作步骤

1. 添加如下代码到你的配置清单：

```
host { "www.bitfieldconsulting.com":
    ip     =&gt; "109.74.195.241",
    target =&gt; "/etc/hosts",
    ensure =&gt; present,
}
```

2. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305716418'

notice: /Stage[main]//Node[cookbook]/Host[www.bitfieldconsultir
com]/ensure: created

info: FileBucket adding /etc/hosts as {md5}977bf5811de978b7f041
9e77b4abe

notice: Finished catalog run in 0.21 seconds
```

工作原理

Puppet 将检查由 target 指定的文件，若主机条目不存在，Puppet 就会添加此条目； 若主机条目存在但 IP 地址不同，Puppet 就会更新此条目。

虽然可以使用有别于 /etc/hosts 的目标文件，但这是默认值，你只需要一个目标文件（即不能使用数组同时指定多个文件）。我认为明确地指定 host 资源的默认值是个很好的做法，因为依托默认的行为会有使代码变得脆弱的倾向。

更多用法

将你的 host 资源组织到类中会对你有帮助。例如，你可以将所有数据库服务器的 host 资源放置在一个名为 admin::dbhosts 的类中，然后在所有的 web 服务器中包含这个类。

一些主机可能需要在多个类中同时定义（例如，一个数据库服务器也可以是一个仓库服务器），使用虚拟资源可以解决这个问题。例如你可以像下面这样，在一个单独的类中将所有的主机定义成虚拟资源：

```
class admin::allhosts
{
    @host { "db1.bitfieldconsulting.com":}
}
```

然后在不同的类中 realize 你需要的主机：

```
class admin::dbhosts
{
    realize( Host["db1.bitfieldconsulting.com"] )
}

class admin::repohosts
{
    realize( Host["db1.bitfieldconsulting.com"] )
}
```

为文件资源指定多个源

Puppet 的 file 资源有一个实用功能，那就是可以为文件指定多个源。 Puppet 会按顺序查找每一个。如果第一个不存在，就继续查找下一个，以此类推。 你可以利用这个功能指定一个默认文件源的替代品， 若一个甚至一系列替代品不存在就会使用默认的（最后一个列出的）文件源。

操作步骤

1. 添加如下的类到你的配置清单：

```
class mysql::app-config( $app )
{
    file { "/etc/my.cnf":
        source =&gt; [ "puppet:///modules/admin/${app}.my.cnf",
                    "puppet:///modules/admin/generic.my.cnf", ]
    }
}
```

2. 使用如下内容创建 /etc/puppet/modules/admin/files/minutespace.my.cnf 文件：

```
# MinuteSpace config file
```

3. 使用如下内容创建 /etc/puppet/modules/admin/files/generic.my.cnf 文件：

```
# Generic config file
```

4. 在一个节点中添加如下代码：

```
class { "mysql::app-config": app =&gt; "minutespace" }
```

5. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305897071'

notice: /Stage[main]/Mysql::App-config/File[/etc/my.cnf]/ensure
defined content as '{md5}24f04b960f4d33c70449fbc4d9f708b6'

notice: Finished catalog run in 0.35 seconds
```

6. 检查 Puppet 是否部署了适用于指定应用程序的 config 文件：

```
# cat /etc/my.cnf
# MinuteSpace config file
```

7. 现在更改节点的定义为：

```
class { "mysql::app-config": app =&gt; "shreddit" }
```

8. 再次运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1305897864'
--- /etc/my.cnf 2011-05-20 13:17:56.006239489 +0000
+++ /tmp/puppet-file20110520-15575-1icobgs-0 2011-05-20
13:24:25.030296062 +0000

@@ -1 +1 @@
-# MinuteSpace config file
+# Generic config file

info: FileBucket adding /etc/my.cnf as {md5}24f04b960f4d33c7044
c4d9f708b6

info: /Stage[main]/Mysql::App-config/File[/etc/
my.cnf]: Filebucketed /etc/my.cnf to puppet with sum
24f04b960f4d33c70449fbc4d9f708b6

notice: /Stage[main]/Mysql::App-config/File[/etc/my.cnf]/conter
content changed '{md5}24f04b960f4d33c70449fbc4d9f708b6' to '{mc
b3a6e744c3ab78dfb20e46ff55f6c33c'

notice: Finished catalog run in 0.93 seconds
```

工作原理

我们定义了 /etc/my.cnf 文件有如下的两个源：

```
file { "/etc/my.cnf":
    source => [ "puppet:///modules/admin/${app}.my.cnf",
                "puppet:///modules/admin/generic.my.cnf", ],
}
```

$app 的值由任何一个使用它的类传递。在第一个例子中，我们为 app 传递了 minutespace：

```
class { "mysql::app-config": app => "minutespace" }
```

Puppet 将首先查找 modules/admin/files/minutespace.my.cnf 文件。 由于此文件存在，所以就会使用它。到目前为止，一切正常。

然后我们把 app 的值更改为 shreddit。Puppet 现在会查找 modules/admin/files/shreddit.my.cnf 文件。 由于此文件不存在，所以 Puppet 试图查找源列表中的下一个文件： modules/admin/files/generic.my.cnf。因为此文件存在，所以会将它部署到节点。

更多用法

你可以在任何一个 file 资源中使用这种手段来处理。例如， 一些节点可能需要针对特定主机的配置，而另一些节点则不需要，你可以使用类似于如下的代码实现：

```
file { "/etc/stuff.cfg":
    source => [ "puppet:///modules/stuff/${hostname}.cfg",
                "puppet:///modules/stuff/generic.cfg" ],
}
```

然后将你的通用配置放在 generic.cfg 文件中。如果主机 cartman 需要一个特殊的配置， 将适用于此主机的配置放在 cartman.cfg 文件中。 cartman.cfg 文件优先于 generic.cfg 文件，因为它在源数组中是首先被列出的。

参见本书

- 第 4 章的 给类传递参数 一节

使用文件资源递归地分发整个目录树

> To understand recursion, you must first understand recursion.
>
> — Saying

当你需要使用 Puppet 分发一批位于同一目录下的多个文件时， 就应该考虑使用文件资源的递归特性来替代逐个文件的分发。 如果你在一个目录上设置了 recurse 参数， Puppet 会复制此目录及其子目录中的所有文件到节点目录，例如：

```
file { "/usr/lib/nagios/plugins/custom":
    source  => "puppet:///modules/nagios/plugins",
    require => Package["nagios-plugins"],
    recurse => true,
}
```

操作步骤

1. 在 Puppet 仓库中的适当位置创建如下的目录树：

   ```
   # mkdir /etc/puppet/modules/admin/files/tree
   # mkdir /etc/puppet/modules/admin/files/tree/a
   # mkdir /etc/puppet/modules/admin/files/tree/b
   # mkdir /etc/puppet/modules/admin/files/tree/c
   # mkdir /etc/puppet/modules/admin/files/tree/a/1
   ```

2. 添加如下代码到你的配置清单：

   ```
   file { "/tmp/tree":
       source  =&gt; "puppet:///modules/admin/tree",
       recurse =&gt; true,
   }
   ```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304768523'

notice: /Stage[main]//Node[cookbook]/File[/tmp/tree]/ensure:
created

notice: /File[/tmp/tree/a]/ensure: created
notice: /File[/tmp/tree/a/1]/ensure: created
notice: /File[/tmp/tree/b]/ensure: created
notice: /File[/tmp/tree/c]/ensure: created
notice: Finished catalog run in 1.25 seconds
```

工作原理

如果一个 file 资源上设置了 recurse 参数，而且此资源又是个目录，那么 Puppet 不仅会部署目录自身，而且还会分发其中的所有内容（包括子目录及其文件）。 对于要在节点上部署目录树及其所有的文件来说，这是一个很好的方法。 也可以用这种方法使用一个单独的资源快速创建大量路径。

更多用法

有时你要部署一些文件到一个已经存在的目录，但是首先要移除该目录下的所有没被 Puppet 管理的文件。 例如，Ubuntu 的 /etc/apt/sources.list.d 目录下的文件，你可能想要确保没有文件存在，即删除那些不是来自 Puppet 的文件。

purge 参数可以为我们实现这个功能。在 Puppet 中定义如下的目录的 file 资源：

```
file { "/etc/apt/sources.list.d":
    ensure  => directory,
    recurse => true,
    purge   => true,
}
```

联合使用 recurse 和 purge 将会移除 /etc/apt/sources.list.d 目录及其子目录下的所有不被 Puppet 管理的文件。 然后你就可以使用独立的 file 资源将自己的文件部署到 /etc/apt/sources.list.d 目录：

```
file { "/etc/apt/sources.list.d/bitfield.list":
    content => "deb http://packages.bitfieldconsulting.com/lucid-ma
}
```

如果有一个子目录，其中包含了你不想 purge 的文件， 可以将此子目录作为 Puppet 的一个 file 资源单独进行定义， 这样这个子目录中的内容就会保留：

```
file { "/etc/exim4/conf.d/acl":
    ensure => directory,
}
```

> 要知道，至少在 Puppet 的当前实现中，采用递归方式复制文件相当慢而且会占用大量服务器的内存。如果要部署的文件不经常改变，使用 tarball 进行部署比使用这种递归方式会更好。

清理过期的旧文件

我们每隔一段时间就会打扫一次房间。 Puppet 的 tidy 资源可以帮助你清理已过期的旧文件，从而减少不必要的磁盘占用。 例如，如果你像 生成报告 一节描述的那样启用了 Puppet 的报告， 就会希望定期清除旧的报告文件。

操作步骤

1. 添加如下代码到你的配置清单：

```
tidy { "/var/lib/puppet/reports":
    age     =&gt; "1w",
    recurse =&gt; true,
}
```

2. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin info: Caching catalog for cookbook.
bitfieldconsulting.com

notice: /Stage[main]//Node[cookbook]/Tidy[/var/lib/puppet/
reports]: Tidying File[/var/lib/puppet/reports/cookbook.
bitfieldconsulting.com/201102241546.yaml]

notice: /Stage[main]//Node[cookbook]/Tidy[/var/lib/puppet/
reports]: Tidying File[/var/lib/puppet/reports/cookbook.
bitfieldconsulting.com/20110214727.yaml]

...

info: Applying configuration version '1306149187'

notice: /File[/var/lib/puppet/reports/cookbook.bitfieldconsulti
com/201102241546.yaml]/ensure: removed

notice: /File[/var/lib/puppet/reports/cookbook.bitfieldconsulti
com/201102141727.yaml]/ensure: removed …

notice: Finished catalog run in 1.48 seconds
```

工作原理

Puppet 会搜索指定路径下所有匹配 age 参数的文件：本例中的存活时间为 1w（一周）。 也可以搜索子目录（recurse => true）。

任何与你指定的标准匹配的文件都会被删除。

更多用法

你可以指定文件存活时间的单位，单位可以是：秒（s）、小时（h）、天（d）和周（w），下面是几个例子：

- 60s
- 180m
- 24h
- 30d
- 4w

你也可以指定文件的大小，当文件的尺寸大于你指定的尺寸后就会删除它，例如：

```
size => "100m",
```

这会删除所有大于 100 megabytes（m） 的文件。单位也可以是：kilobytes（k）或 bytes（b）。

> 值得注意的是，如果你同时指定了 age 和 size 参数，它们将被视为独立的标准（即参数之间是“或”逻辑而不是“与”逻辑）。例如，你对一个目录的文件指定了如下的参数，那么 Puppet 会删除所有超过一天的文件，同时也会删除所有尺寸大于 512KB 的文件。
>
> ```
> age => "1d",
> size => "512k",
> ```

使用日程表资源

使用启用了 schedule 参数的资源，你可以控制当前资源何时被应用。 例如，你希望如下的 exec 资源每天应用一次，将资源参数 schedule 设置成了内置的值 daily：

```
exec { "/usr/bin/apt-get update":
    schedule => daily,
}
```

遗憾的是，给 schedule 参数指定 daily 并不能保证该资源每天都能应用一次。 内置的 daily 仅能限制 exec 资源在一天之内不能被应用多余一次， 但是资源是否被应用以及何时被应用完全取决于 Puppet 是否运行以及何时运行。

正因为如此，使用 schedule 资源才是安排执行其他资源的最佳选择。例如， 你可能想要确保 apt-get update 一小时内运行不超过一次； 或者确保一项维护工作在白天生产时间期间不被执行。

为此，你需要创建自己的 schedule 资源。

操作步骤

1. 添加如下代码到你的配置清单：

   ```
   schedule { "not-in-office-hours":
       period =&gt; daily,
       range  =&gt; [ "17:00-23:59", "00:00-09:00" ],
       repeat =&gt; 1,
   }

   exec { "/bin/echo Doing maintenance!":
       schedule =&gt; "not-in-office-hours",
   }
   ```

2. Run Puppet。

工作原理

我们创建了名为 not-in-office-hours 的 schedule，它指定了重复周期为 daily, 且指定了允许的时间范围为下午 5 点到次日早上 9 点：

```
    period  => daily,
    range   => [ "17:00-23:59", "00:00-09:00" ],
```

我们同时设置了在每个时间周期内资源被应用的最大次数为 1：

```
repeat => 1,
```

然后，我们在如下的 exec 资源中使用这个自定义的 schedule：

```
exec { "/bin/echo Doing maintenance!":
    schedule => "not-in-office-hours",
}
```

若 exec 资源不使用 schedule 参数，每次运行 Puppet，这个 exec 资源就会被应用一次。 现在，Puppet 将依照 not-in-office-hours 资源的设置做如下测试：

- 当前时间是否在允许的范围内
- 在一个重复周期内，资源是否已经应用到了允许的最大次数

例如，让我考察在如下连续的几个小时内如果 Puppet 每小时执行一次将会发生什么：

- 4 p.m.：超出了允许的时间范围，所以 Puppet 不会做任何事情。
- 5 p.m.：在允许的时间范围内，且在此重复周期内还没有运行过，所以 Puppet 会应用此资源。
- 6 p.m.：在允许的时间范围内，且在此重复周期内已经运行过一次，由于已经到达了 repeat 设置的最大值，所以 Puppet 不会做任何事情。

依此类推，直到第二天再重复这一过程。

更多用法

如果需要，你可以增大 repeat 参数的值，例如：在每小时内运行一个作业不超过 6 次：

```
period => hourly,
repeat => 6,
```

要记住这不能保证每个小时此作业都会运行 6 次。这只是设置了一个上限。 无论 Puppet 经常运行还是会发生其他情况，如果一个小时内已经运行了 6 次就不会再次运行。 如果 Puppet 每天仅执行一次，那么这个作业也仅执行一次。 因此，要确保某事件在特定的时间 不会 发生（或者 不 超过给定的频率），使用 schedule 是最有用的。

资源的审计

不是每个问题都有技术性的答案。我曾经诊断过一台服务器，它对 ping、SSH 或控制台连接均无响应。 我不能确定这到底是硬件故障还是软件故障。

当我电话询问主机所在位置的站点客服时，奥秘最终被揭开了。他们告诉我： 先期抵达的两名不明身份的男子，进入大厦直奔服务器机房，拔下了机器，并轻易地将机器带出了大厦。 后来我们发现，在机房所在的地区（Missouri，M.O.）曾发生过一连串的电脑失窃案。

从这个案例获得的重要信息是：最终知道了谁对你的服务器做了些什么，这还是不错的。

模拟运行模式（使用 --noop 开关）审计由 Puppet 控制的任何机器的变化是一种简单的方法。 然而，Puppet 还有一个专门的审计功能，它可以报告资源或特定属性的变化。

操作步骤

使用 audit 元参数定义一个资源：

```
file { "/etc/passwd":
    audit => [ owner, mode ],
}
```

工作原理

元参数（**metaparameter**）是一种可以应用到任何资源的参数，不仅用于指定类型。 audit 元参数告诉 Puppet，你想要记录和监视关于这个资源的某些事件。 其值可以是你要审计的一个参数列表。

在本例中，一旦 Puppet 运行，它就开始记录 /etc/passwd 文件的属主和权限模式。 无论是属主改变还是权限改变，例如：如果你运行了如下命令：

```
# chmod 666 /etc/passwd
```

当下一次运行 Puppet 时，它将会察觉到这个变化并将其记入日志：

```
notice: /Stage[main]//Node[cookbook]/File[/etc/passwd]/mode: audit
change: previously recorded value 644 has been changed to 666
```

更多用法

对于需要审记大型网络中的机器所做的任何改变（无论是恶意的还是意外的）的情况， 使用 audit 特性是非常有用的。 你还可以使用 tagmail 报告特性通过 e-mail 自动发送审计通知信息。

对于要时刻关注不被 Puppet 管理的任何事物（例如生产服务器上的应用程序代码）的情况， 使用 audit 特性也是非常方便的。 你可以从 http://www.puppetlabs.com/blog/allabout-auditing-with-puppet/ 获取更多关于 Puppet 审计能力的信息。

如果你希望审计一个资源的一切，可以使用如下的代码：

```
file { "/etc/passwd":
    audit => all,
}
```

参见本书

- 第 2 章的 测试你的 Puppet 配置清单 一节
- 第 2 章的 通过 Email 发送包含特定标签的日志信息 一节

临时禁用资源

有时候，你希望在不会干扰其他工作的情况下临时禁用一个资源。例如， 你可能想要调整服务器上的一个配置文件，经过 Puppet 的不断测试，直到获得你想要的确切设置。 在此期间你不想让 Puppet 覆盖旧版本的配置，为此你可以在资源上设置如下的 noop 元参数：

```
    noop => true,
```

操作步骤

1. 添加如下代码到你的配置清单：

```
file { "/tmp/test.cfg":
    content =&gt; "Hello, world!\n",
    noop =&gt; true,
}
```

2. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1306159566'

notice: /Stage[main]//Node[cookbook]/File[/tmp/test.cfg]/ensure
is absent, should be file (noop)

notice: Finished catalog run in 0.53 seconds
```

3. 测试之后，移除 noop 参数：

```
file { "/tmp/test.cfg":
    content =&gt; "Hello, world!\n",
}
```

4. 再次运行 Puppet：

```
# puppet agent --test

info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1306159705'

notice: /Stage[main]//Node[cookbook]/File[/tmp/test.cfg]/ensure
defined content as '{md5}746308829575e17c3331bbcb00c0898b'

notice: Finished catalog run in 0.52 seconds
```

工作原理

首次运行 Puppet 时，由于 noop 元参数被设置为 true， 因此对于这一特定资源，与你使用 --noop 标志运行 Puppet 的效果是一样的。 Puppet 会指出此资源将被应用， 除此之外什么也没做。

再次运行 Puppet 时，由于 noop 元参数被移除，所以此资源会被正常地应用到节点。

管理时区

> I try to take one day at a time, but sometimes several days attack at once.
>
> — Ashleigh Brilliant

你迟早会遭遇由于服务器的时区不同而带来的各种奇怪问题。 为了避免此类问题的发生，确保所有服务器都使用相同的时区是明智之举。 无论这些服务器所处的地理位置在何处，都应该使用相同的时区（GMT 是合乎逻辑的选择）。

除非服务器是由太阳能供电的，否则我认为没有任何理由不关心服务器的时区设置。

操作步骤

1. 使用如下内容创建 /etc/puppet/modules/admin/manifests/gmt.pp 文件：

```
class admin::gmt
{
    file { "/etc/localtime":
        ensure =&gt; link,
        target =&gt; "/usr/share/zoneinfo/GMT",
    }
}
```

2. 在所有的节点中添加如下代码：

```
include admin::gmt
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1304955158'

info: FileBucket adding /etc/localtime as {md5}02b73b0cf0d96e2f
ae56b178bf58e

info: /Stage[main]/Admin::Gmt/File[/etc/localtime]: Filebuckete
etc/localtime to puppet with sum 02b73b0cf0d96e2f75cae56b178bf5

notice: /Stage[main]/Admin::Gmt/File[/etc/localtime]/ensure:
ensure changed 'file' to 'link'

notice: Finished catalog run in 1.94 seconds
```

更多用法

如果你想要使用其他的时区，请在 /usr/share/zoneinfo 中选择相应的文件，例如：US/Eastern。

应用程序

> The best software in the world only sucks. The worst software is significantly worse than that.
>
> — Luke Kanies

在本章中，我们将学习如下内容：

- 管理 Apache 服务
- 创建 Apache 虚拟主机
- 创建 Nginx 虚拟主机
- 创建 MySQL 数据库及用户
- 管理 Drupal 站点
- 管理 Rails 应用程序

如果没有应用程序，服务器仅仅是一个非常昂贵的空间加热器。 本章将介绍一些使用 Puppet 管理一些特定应用程序的处方：MySQL、Apache、Nginx、Rails 和 Drupal。 这些都是非常流行的应用程序， 所以这对你会很有用。 然而，它们使用的模式和技术几乎适用于任何软件， 所以适当改写这些处方以适应自己的目的并非难事。

管理 Apache 服务

Apache 是一款流行的 web 服务器，然而对于其配置者来说并不轻松。 Puppet 可以在一定程度上缓解配置者因管理 Apache 服务器所带来的痛苦。

操作步骤

1. 如果 Apache 模块还不存在就创建它：

```
# mkdir /etc/puppet/modules/apache
# mkdir /etc/puppet/modules/apache/templates
# mkdir /etc/puppet/modules/apache/manifests
```

2. 使用如下内容创建 /etc/puppet/modules/apache/manifests/init.pp 文件：

```
class apache {
    package { "apache2-mpm-prefork": ensure =&gt; installed }

    service { "apache2":
        enable  =&gt; true,
        ensure  =&gt; running,
        require =&gt; Package["apache2-mpm-prefork"],
    }

    file { "/etc/apache2/logs":
        ensure  =&gt; directory,
        require =&gt; Package["apache2-mpm-prefork"],
    }

    file { "/etc/apache2/conf.d/name-based-vhosts.conf":
        content =&gt; "NameVirtualHost *:80",
        require =&gt; Package["apache2-mpm-prefork"],
        notify  =&gt; Service["apache2"],
    }
}
```

3. 在一个节点中添加如下代码：

```
include apache
```

4. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1309189590'

notice: /Stage[main]/Apache/Package[apache2-mpm-prefork]/ensure
ensure changed 'purged' to 'present'

notice: /Stage[main]/Apache/File[/etc/apache2/logs]/ensure:
created

notice: /Stage[main]/Apache/File[/etc/apache2/conf.d/name-based
conf]/ensure: defined content as '{md5}78465aacbd01eb537b94
1b21ae0af8b8'

info: /Stage[main]/Apache/File[/etc/apache2/conf.d/name-basedvh
conf]: Scheduling refresh of Service[apache2]

notice: Finished catalog run in 39.45 seconds
```

更多用法

在下一节中，我们将看到如何为 Apache 定义虚拟主机的内容。 然而你可能会发现，你需要为整个 Apache 服务器（包括虚拟主机）提供特殊的配置选项。 你可以使用 Puppet 通过部署 apache2.conf 来配置这些选项， 但是将配置片段放在 /etc/apache2/conf.d 目录中会更加整洁清晰。 例如，你可以在 init.pp 中添加如下代码：

```
define snippet() {
    file { "/etc/apache2/conf.d/${name}":
        source => "puppet:///modules/apache/${name}",
        notify => Service["apache2"],
    }
}
```

并在节点上包含如下的代码片段：

```
apache::snippet { "site-specific.conf": }
```

创建 Apache 虚拟主机

使用 ERB 模板配置虚拟主机是一种常见的应用， 因为每个虚拟主机配置的实例通常都使用类似的样板代码，只有一两个变量的值不同而已。 显然，对于某些网站或应用程序来说，你需要在虚拟主机的定义中指定特殊的配置选项， 然而这些特殊选项又不能通过一个简单的模板来配置?—?但是，不管怎样， 使用一个模板配置一些简单的站点将会节省时间、避免重复劳动。

操作步骤

1. 添加如下代码到 /etc/puppet/modules/apache/manifests/init.pp：

```
define site( $sitedomain = "", $documentroot = "" ) {
    include apache

    if $sitedomain == "" {
        $vhost_domain = $name
    } else {
        $vhost_domain = $sitedomain
    }

    if $documentroot == "" {
        $vhost_root = "/var/www/${name}"
    } else {
        $vhost_root = $documentroot
    }

    file { "/etc/apache2/sites-available/${vhost_domain}.conf":
        content =&gt; template("apache/vhost.erb"),
        require =&gt; File["/etc/apache2/conf.d/name-basedvhost
        notify =&gt; Exec["enable-${vhost_domain}-vhost"],
    }

    exec { "enable-${vhost_domain}-vhost":
        command     =&gt; "/usr/sbin/a2ensite ${vhost_domain}.c
        require     =&gt; [ File["/etc/apache2/sites-available/
 vhost_domain}.conf"], Package["apache2-mpm-prefork"] ],
        refreshonly =&gt; true,
        notify      =&gt; Service["apache2"],
    }
}
```

2. 使用如下内容创建 /etc/puppet/modules/apache/templates/vhost.erb 文件：

```
&lt;VirtualHost *:80&gt;
    ServerName &lt;%= vhost_domain %&gt;
    ServerAdmin admin@&lt;%= vhost_domain %&gt;
    DocumentRoot &lt;%= vhost_root %&gt;
    ErrorLog logs/&lt;%= vhost_domain %&gt;-error_log
    CustomLog logs/&lt;%= vhost_domain %&gt;-access_log common

    &lt;Directory /var/www/&lt;%= vhost_domain %&gt;&gt;
        Allow from all
        Options +Includes +Indexes +FollowSymLinks
        AllowOverride all
    &lt;/Directory&gt;
&lt;/VirtualHost&gt;

&lt;VirtualHost *:80&gt;
    ServerName www.&lt;%= vhost_domain %&gt;
    Redirect 301 / http://&lt;%= vhost_domain %&gt;/
&lt;/VirtualHost&gt;
```

3. 添加如下代码到一个节点：

```
apache::site { "keithlard.com": }
```

4. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1309190720'

notice: /Stage[main]//Node[cookbook]/Apache::Site[keithlard.com
File[/etc/apache2/sites-available/keithlard.com.conf]/ensure:
defined content as '{md5}f2a558c02beeaed4beb7da250821b663'

info: /Stage[main]//Node[cookbook]/Apache::Site[keithlard.com]/
File[/etc/apache2/sites-available/keithlard.com.conf]: Scheduli
refresh of Exec[enable-keithlard.com-vhost]

notice: /Stage[main]//Node[cookbook]/Apache::Site[keithlard.com
Exec[enable-keithlard.com-vhost]: Triggered 'refresh' from 1
events

info: /Stage[main]//Node[cookbook]/Apache::Site[keithlard.
com]/Exec[enable-keithlard.com-vhost]: Scheduling refresh of
Service[apache2]

notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refres
from 2 events
notice: Finished catalog run in 3.79 seconds
```

工作原理

名为 apache::site 的 define 使用 vhost.erb 模板生成 Apache 虚拟主机的定义。 默认情况下，假设站点的域名与站点实例的名字相同，本例中是 keithlard.com。 所以当 Puppet 看到如下代码时：

```
apache::site { "keithlard.com": }
```

它就使用 keithlard.com 作为站点域名。如果你要指定不同的域名，请添加 sitedomain 参数：

```
apache::site { "networkr_production":
    sitedomain => "networkr.com",
}

apache::site { "networkr_staging":
    sitedomain => "staging.networkr.com",
}
```

模板系统的优秀之处在于：如果你想为所有站点重新配置一个值（例如，更改管理员的 e-mail 地址），你只需要修改一次模板，Puppet 就会根据模板相应地更新所有的虚拟主机。

同样地，如果你需要为虚拟主机指定与默认值（/var/www/${name}）不同的 DocumentRoot，请添加如下的 documentroot 参数：

```
apache::site { "communitysafety.org":
    documentroot => "/var/apps/commsafe",
}
```

更多用法

在前面的例子中，我们只在模板中定义了一个变量，但只要你愿意，你可以使用更多的变量。 它们也可以是 **facts**，例如：

```
ServerName <%= fqdn %>
```

或者 Ruby 表达式：

```
ServerAdmin<%= emails["admin"] %>
```

或者任何你要执行的 Ruby 代码：

```
ServerAdmin <%= vhost_domain == 'coldcomfort.com' ? 'seth@coldcomf
com' : 'flora@poste.com' %>
```

参见本书

- 第 5 章的 在模板中遍历数组 一节

创建 Nginx 虚拟主机

Nginx 是一个快速的、轻量级的 Web 服务器软件， 在许多情况下它已取代了 Apache，尤其是运行 Web 应用程序的情况。 然而，其配置语言与 Apache 相比并没有做什么重大改进，仍然不够简单清晰。 此外，大多数的配置文档只有俄语版，这也就解释了为什么你看到这么多份关于 "Understanding Russian for Nginx Administrators" 的拷贝。

准备工作

你需要 当文件更新时运行命令 一节中曾使用的 Nginx 模块。 如果你像本章 管理 Apache 服务 一节中那样创建了 Apache 模块， 你还需要使用如下命令关闭 Apache 服务：

```
# service apache2 stop
```

操作步骤

1. 添加如下代码到 /etc/puppet/modules/nginx/manifests/init.pp：

```
define site( $sitedomain = "" ) {
    include nginx

    if $sitedomain == "" {
        $vhost_domain = $name
    } else {
        $vhost_domain = $sitedomain
    }

    file { "/etc/nginx/sites-available/${vhost_domain}.conf":
        content =&gt; template("nginx/vhost.erb"),
        require =&gt; Package["nginx"],
    }

    file { "/etc/nginx/sites-enabled/${vhost_domain}.conf":
        ensure  =&gt; link,
        target  =&gt; "/etc/nginx/sites-available/${vhost_domai
 conf",
        require =&gt; File["/etc/nginx/sites-available/${vhost_
 domain}.conf"],
        notify  =&gt; Exec["reload nginx"],
    }
}
```

2. 使用如下内容创建 /etc/puppet/modules/nginx/templates/vhost.erb 文件：

```
server {
    listen 80;
    server_name &lt;%= vhost_domain %&gt;;
    access_log /var/log/nginx/&lt;%= vhost_domain %&gt;-access_
    root /var/www/&lt;%= vhost_domain %&gt;;
}
```

3. 创建目录 /var/www/bbqrecipes.com，使用适当的信息创建此目录下的 index.html 文件：

```
Welcome to the BBQ Recipes site!
```

4. 添加如下代码到一个节点：

```
nginx::site { "bbqrecipes.com": }
```

5. 运行 Puppet：

```
# puppet agent -test

info: Retrieving plugin info: Caching catalog for cookbook.
bitfieldconsulting.com info: Applying configuration version
'1309198476'

notice: /Stage[main]/Nginx/Package[nginx]/ensure: ensure change
'purged' to 'present'

notice: /Stage[main]//Node[cookbook]/Nginx::Site[bbqrecipes.com
File[/etc/nginx/sites-available/bbqrecipes.com.conf]/ensure:
defined content as '{md5}fa92d2e7543b378e26827a063be34a31'

notice: /Stage[main]//Node[cookbook]/Nginx::Site[bbqrecipes.com
File[/etc/nginx/sites-enabled/bbqrecipes.com.conf]/ensure: crea

info: /Stage[main]//Node[cookbook]/Nginx::Site[bbqrecipes.com]/
File[/etc/nginx/sites-enabled/bbqrecipes.com]: Scheduling refre
of Exec[reload nginx]

notice: /Stage[main]/Nginx/Service[nginx]/ensure: ensure change
'stopped' to 'running'

notice: /Stage[main]/Nginx/Exec[reload nginx]: Triggered 'refre
from 1 event

notice: Finished catalog run in 21.45 seconds
```

工作原理

以 bbqrecipes.com 为资源名对名为 nginx::site 的 define 创建一个实例， Puppet 会使用变量 vhost_domain 的值 bbqrecipes.com 调用 vhost.erb 模板生成相应的配置文件 bbqrecipes.com.conf。 此文件包含了 Nginx 为了响应对 domain 的请求所需要知道的一切， 并且指明了与 domain 对应的根文档目录所映射的文件系统位置。

更多用法

与 Apache 不同，Nginx 直到现在还不支持动态模块。这就意味着， 如果想为其添加默认不包含的特殊功能，你必须自己重新编译 Nginx。 正确的做法是，使用你想要的选项编译 Nginx，然后创建一个软件包，将其放在自己的仓库中 （参见 配置 APT 软件仓库 一节的描述）提供给需要它的节点使用。

然而，一些使用 Puppet 的系统管理员希望省略这个步骤，直接提供源码包并在目标服务器上编译。 为了实现这点，可以使用 exec 资源通过类似于 从源码包自动构建软件 一节中描述的模式实现。 对于敏捷开发环境而言，被管理的有关产品几乎每隔几天就会有所变化，在这种情况下， 使用这种做法比不断重构软件包要迅速且节省由于重构软件包带来的成本。

参见本书

- 本章的 管理 Rails 应用程序 一节

创建 MySQL 数据库及用户

MySQL 是一个使用广泛的数据库服务器，你肯定会在某些节点上安装配置 MySQL 服务器。 本节将向你展示如何安装配置 MySQL 服务器，以及如何为应用程序自动创建数据库和用户。

准备工作

1. 如果你还没有 MySQL 模块，先创建一个：

```
# mkdir /etc/puppet/modules/mysql
# mkdir /etc/puppet/modules/manifests
# mkdir /etc/puppet/modules/files
```

2. 使用如下内容创建 /etc/puppet/modules/mysql/manifests/server.pp 文件：

```
class mysql::server {
    package { "mysql-server": ensure =&gt; installed }

    service { "mysql":
        enable =&gt; true,
        ensure =&gt; running,
        require =&gt; Package["mysql-server"],
    }

    file { "/etc/mysql/my.cnf":
        owner =&gt; "mysql", group =&gt; "mysql",
        source =&gt; "puppet:///mysql/my.cnf",
        notify =&gt; Service["mysql"],
        require =&gt; Package["mysql-server"],
    }

    exec { "set-mysql-password":
        unless =&gt; "/usr/bin/mysqladmin -uroot -p${mysql_pass
 status",
        command =&gt; "/usr/bin/mysqladmin -uroot password ${my
 password}",
        require =&gt; Service["mysql"],
    }
}
```

3. 使用如下内容创建 /etc/puppet/modules/mysql/files/my.cnf 文件：

```
[client]
port = 3306
socket = /var/run/mysqld/mysqld.sock

[mysqld_safe]
socket = /var/run/mysqld/mysqld.sock
nice = 0

[mysqld]
user = mysql
socket = /var/run/mysqld/mysqld.sock
port = 3306
datadir = /var/lib/mysql

!includedir /etc/mysql/conf.d/
```

4. 添加如下代码到 /etc/puppet/manifests/site.pp 文件：

```
$mysql_password = "secret"
```

5. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1309448283'

notice: /Stage[main]/Mysql::Server/Package[mysql-server]/ensure
ensure changed 'purged' to 'present'

notice: /Stage[main]/Mysql::Server/File[/etc/mysql/my.cnf]/owne
owner changed 'root' to 'mysql'

notice: /Stage[main]/Mysql::Server/File[/etc/mysql/my.cnf]/grou
group changed 'root' to 'mysql'

info: /Stage[main]/Mysql::Server/File[/etc/mysql/my.cnf]:
Scheduling refresh of Service[mysql]

info: /Stage[main]/Mysql::Server/File[/etc/mysql/my.cnf]:
Scheduling refresh of Service[mysql]

notice: /Stage[main]/Mysql::Server/Service[mysql]/enable: enabl
changed 'false' to 'true'

notice: /Stage[main]/Mysql::Server/Service[mysql]: Triggered
'refresh' from 2 events

notice: Finished catalog run in 61.78 seconds
```

操作步骤

1. 添加如下代码到 /etc/puppet/modules/mysql/manifests/server.pp 文件：

```
define db( $user, $password ) {
    include mysql::server

    exec { "create-${name}-db":
        unless  =&gt; "/usr/bin/mysql -u${user} -p${password}${
        command =&gt; "/usr/bin/mysql -uroot -p${mysql_passworc
  \"create database ${name}; grant all on ${name}.* to
  ${user}@localhost identified by '$password'; flush
 privileges;\"",
        require =&gt; Service["mysql"],
    }
}
```

2. 添加如下代码到一个节点：

```
mysql::server::db { "johnstest":
    user =&gt; "john",
    password =&gt; "johnstest",
}
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1309449259'

notice: /Stage[main]//Node[cookbook]/Mysql::Server::Db[johnstes
Exec[create-johnstest-db]/returns: executed successfully

notice: Finished catalog run in 1.61 seconds
```

4. 检查数据库是否已经创建，以及用户和权限的正确性：

```
# mysql -ujohn -pjohnstest johnstest

Reading table information for completion of table and column na
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 36
Server version: 5.1.41-3ubuntu12.10 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.

mysql&gt;
```

工作原理

mysql::server 类安装 MySQL，并使用你在 site.pp 文件中设置的 root 用户口令配置 MySQL。 名为 mysql::server::db 的 define 允许我们使用一个指定的名字创建数据库， 以及一个能访问此数据库的相关的 MySQL 用户 。 例如，一个典型的 web 应用程序可能需要一个以应用程序命名的数据库， 以及一个可以登录数据库的特定用户名。

更多用法

要创建多个数据库，只需添加多个 mysql::server::db 实例：

```
mysql::server::db { [ "test1", "test2", "test3" ]:
    user => "john",
    password => "johnstest",
}
```

管理 Drupal 站点

Drupal 是一个内容管理系统，它通过插拔组装一系列罐装的模块让你快速构建网站, 它使用户创建和编辑自己的内容变的相对容易。 Drupal 特别适合使用 Puppet 来管理，因为有一个强大的命令行工具 drush， 你可以使用这个工具安装、管理 Drupal 站点。

如果我们将自动化管理 Drupal 站点的 drush 工具与我们已经创建的用于管理 MySQL 数据库和 Apache 虚拟主机的 Puppet 处方相结合, 就可以使用单一资源创建一个安装 Drupal 站点所需一切的新处方。

准备工作

1. 创建一个新的 drupal 模块如下：

```
# mkdir /etc/puppet/modules/drupal
# mkdir /etc/puppet/modules/drupal/manifests
```

2. 使用如下内容创建 /etc/puppet/modules/drupal/manifests/init.pp 文件：

```
class drupal {
    $drupalversion = "7.2"

    exec { "download-drush":
        cwd =&gt; "/root",
        command =&gt; "/usr/bin/wget http://ftp.drupal.org/file
 projects/drush-7.x-4.4.tar.gz ",
        creates =&gt; "/root/drush-7.x-4.4.tar.gz",
        require =&gt; Package["php5-mysql"],
    }

    exec { "install-drush":
        cwd     =&gt; "/usr/local",
        command =&gt; "/bin/tar xvzf /root/drush-7.x-4.4.tar.gz
        creates =&gt; "/usr/local/drush",
        require =&gt; Exec["download-drush"],
    }

    file { "/usr/local/bin/drush":
        ensure =&gt; link,
        target =&gt; "/usr/local/drush/drush",
        require =&gt; Exec["install-drush"],
    }

    exec { "install-drupal":
        cwd     =&gt; "/var/www",
        command =&gt; "/usr/local/drush/drush dl drupal-
  ${drupalversion}",
        creates =&gt; "/var/www/drupal-${drupalversion}",
        require =&gt; Exec["install-drush"],
    }

    file { "/var/www/drupal":
        ensure =&gt; link,
        target =&gt; "/var/www/drupal-${drupalversion}",
        require =&gt; Exec["install-drupal"],
    }

    package { [ "libapache2-mod-php5",
                "php5-mysql" ]: ensure =&gt; installed }

    exec { "enable-mod-php5":
        command =&gt; "/usr/bin/a2enmod php5",
        creates =&gt; "/etc/apache2/mods-enabled/php5.conf",
        require =&gt; Package["libapache2-mod-php5"],
    }
}
```

操作步骤

1. 在 init.pp 文件的 drupal 类中添加如下内容：

```
define site( $password, $sitedomain = "" ) {
    include drupal

    if $sitedomain == "" {
        $drupal_domain = $name
    } else {
        $drupal_domain = $sitedomain
    }

    $dbname = regsubst( $drupal_domain, "\.", "" )
    mysql::server::db { $dbname:
        user =&gt; $dbname,
        password =&gt; $password,
    }

    exec { "site-install-${name}":
        cwd =&gt; "/var/www/drupal",
        command =&gt; "/usr/local/bin/drush site-install -y
 --site-name=${name} --sites-subdir=${drupal_domain}
 --db-url=mysql://${dbname}:${password}@localhost/${dbname}",
        creates =&gt; "/var/www/drupal/sites/${drupal_domain}",
        require =&gt; [ File["/var/www/drupal"], Exec["install-
         Mysql::Server::Db[$dbname] ],
        logoutput =&gt; on_failure,
    }

    apache::site { $drupal_domain:
        documentroot =&gt; "/var/www/drupal",
    }
}
```

2. 添加如下内容到一个节点：

```
drupal::site { "crispinfo.com":
    password =&gt; "crunch",
}
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1309783783'

notice: /Stage[main]//Node[cookbook]/Drupal::Site[crispinfo.com
Mysql::Server::Db[crispinfocom]/Exec[create-crispinfocom-db]/
returns: executed successfully

notice: /Stage[main]//Node[cookbook]/Drupal::Site[crispinfo.com
Apache::Site[crispinfo.com]/File[/etc/apache2/sites-available/
crispinfo.com.conf]/ensure: defined content as '{md5}15c5bbffa6
fce0b8a3996914af549'

info: /Stage[main]//Node[cookbook]/Drupal::Site[crispinfo.com]/
Apache::Site[crispinfo.com]/File[/etc/apache2/sites-available/
crispinfo.com.conf]: Scheduling refresh of Exec[enable-crispinf
com-vhost]

notice: /Stage[main]//Node[cookbook]/Drupal::Site[crispinfo.com
Apache::Site[crispinfo.com]/Exec[enable-crispinfo.com-vhost]:
Triggered 'refresh' from 1 events

info: /Stage[main]//Node[cookbook]/Drupal::Site[crispinfo.com]/
Apache::Site[crispinfo.com]/Exec[enable-crispinfo.com-vhost]:
Scheduling refresh of Service[apache2]

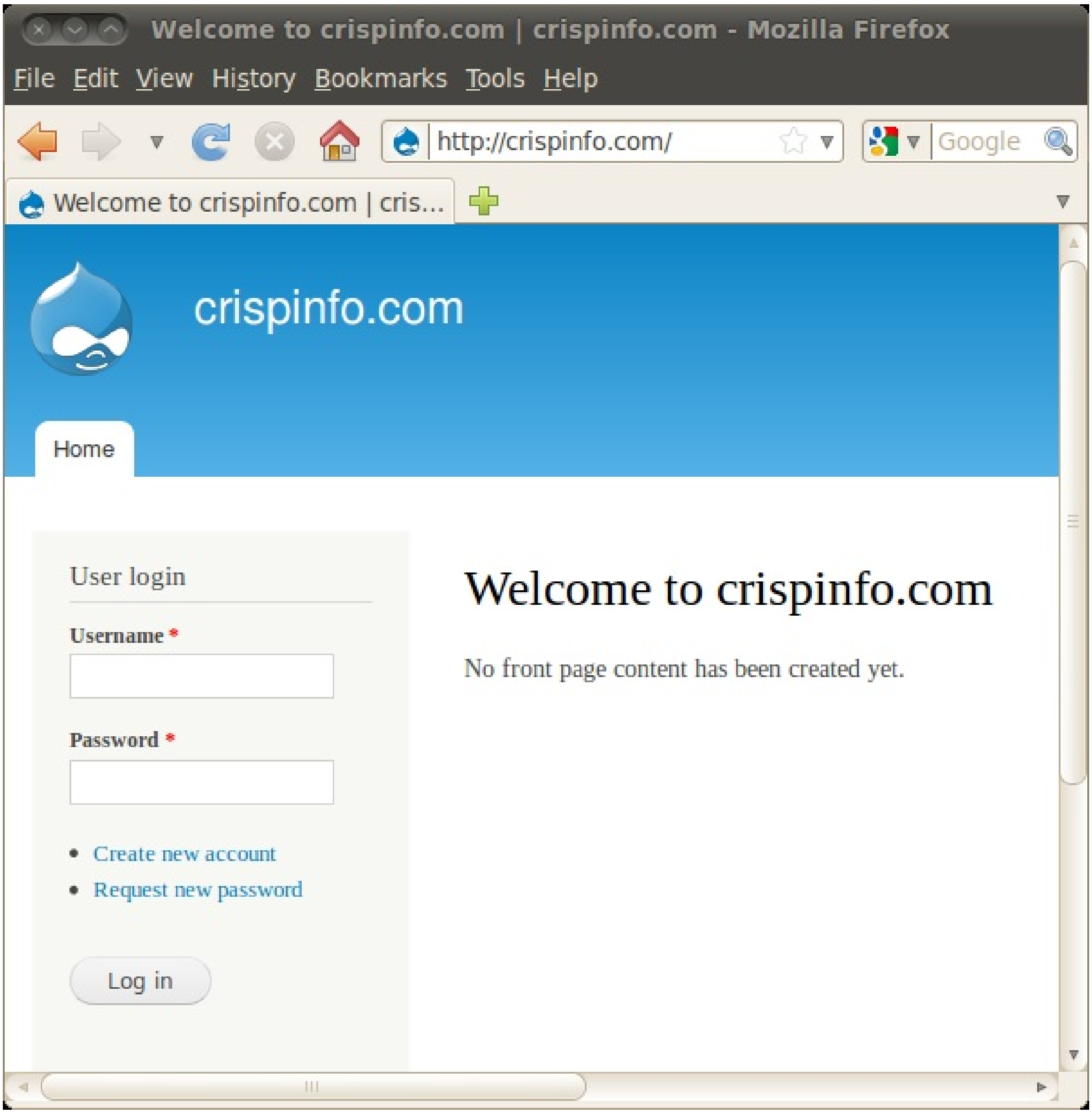
notice: /Stage[main]/Apache/Service[apache2]: Triggered 'refres
from 1 events

notice: /Stage[main]//Node[cookbook]/Drupal::Site[crispinfo.com
Exec[site-install-crispinfo.com]/returns: executed successfully
notice: Finished catalog run in 22.51 seconds
```

4. 在 /etc/hosts 文件中创建一个条目将 crispinfo.com 指向你正在使用的节点 IP（如果还没设置 DNS）：

```
10.0.2.15 crispinfo.com
```

5. 在浏览器中检查站点，以确保一切都已经正确创建。你应该看到 Drupal 的登录提示，在下图所示：



使用由 drush site-install 创建的默认的管理员登录，用户名为 admin 其口令为 admin。 显然你应该为实际生产线上的站点设置强壮的口令 （查看 drush 文档获得如何使用命令行工具设置的信息）。

工作原理

真是神奇呀！尤其是 drupal 类首先安装 drush，然后使用它安装 Drupal 的核心代码 （你可以通过修改 $drupalversion 的值改变版本）。

drupal::sitedefine 为你想要创建的每个站点运行 drush site-install 。 在我们的例子中，创建了一个名为 crispinfo.com 的站点并为其传递了站点数据库使用的口令，其余的工作都由 drush 去完成。

drupal::site 也为我们的站点创建了所需的 **Apache** 虚拟主机 （使用本章 创建 Apache 虚拟主机 一节中的处方) 和 **MySQL** 数据库 （使用本章 创建 MySQL 数据库及用户 一节中的处方）。

更多用法

要管理 Drupal 站点，drush 可以帮你做很多事，包括更新 Drupal 的核心代码、 安装模块和主题模板、管理用户以及备份站点数据库等。 你可以在 http://drush.ws/ 找到更多关于 drush 的信息。

管理 **Rails** 应用程序

Rails 是一个非常受欢迎的 Web 应用程序框架（从某种意义上说，是由于它被广泛应用而不是人们眞正喜欢它）。 因此，在某些时候，你可能会被要求管理它。 本节要介绍的处方包含了安装一台运行 Rails 应用程序服务器所要做的绝大部分工作。 本处方假定你会使用 Nginx 和 **Passenger** 作为 Web 服务器， 然而你也可以轻松地修改本处方，使用 Apache 替换它。

操作步骤

1. 创建 rails 模块的目录结构：

```
# mkdir /etc/puppet/modules/rails
# mkdir /etc/puppet/modules/rails/manifests
# mkdir /etc/puppet/modules/rails/templates
# mkdir /etc/puppet/modules/rails/files
```

2. 使用如下内容创建 /etc/puppet/modules/rails/manifests/init.pp 文件：

```
class rails {
    include rails::passenger

    package { "bundler":
        provider =&gt; gem,
        ensure   =&gt; installed,
    }

    define app( $sitedomain ) {
        include rails

        file { "/opt/nginx/sites-available/${name}.conf":
          content =&gt; template("rails/app.conf.erb"),
          require =&gt; File["/opt/nginx/sites-available"],
        }

        file { "/opt/nginx/sites-enabled/${name}.conf":
          ensure  =&gt; link,
          target  =&gt; "/opt/nginx/sites-available/${name}.cor
          require =&gt; File["/opt/nginx/sites-enabled"],
          notify  =&gt; Exec["reload-nginx"],
        }

        file { "/opt/nginx/conf/includes/${name}.conf":
          source =&gt; [ "puppet:///modules/rails/${name}.conf"
                      "puppet:///modules/rails/empty.conf" ],
          notify =&gt; Exec["reload-nginx"],
        }

        file { [ "/var/www",
                 "/var/www/${name}",
                 "/var/www/${name}/releases",
                 "/var/www/${name}/shared",
                 "/var/www/${name}/shared/config",
                 "/var/www/${name}/shared/log",
                 "/var/www/${name}/shared/system" ]:
            ensure =&gt; directory,
            mode   =&gt; 775,
            owner  =&gt; "www-data",
            group  =&gt; "www-data",
        }
    }
}
```

3. 使用如下内容创建 /etc/puppet/modules/rails/manifests/passenger.pp 文件：

```
class rails::passenger {
    $passenger_version = "3.0.7"
    $passenger_dependencies = [ "build-essential",
                                "libcurl4-openssl-dev",
```

```
                                  "libssl-dev",
                                  "ruby",
                                  "rubygems" ]

   package { $passenger_dependencies: ensure =&gt; installed }
       exec { "install-passenger":
       command =&gt; "/usr/bin/gem install passenger
--version=${passenger_version}",
        unless =&gt; "/usr/bin/gem list | /bin/grep passenger
grep ${passenger_version}",
       require =&gt; [ Package["rubygems"], Package[$passenger
        dependencies] ],
       timeout =&gt; "-1",
   }

   exec { "install-passenger-nginx-module":
       command =&gt; "/usr/lib/ruby/gems/1.8/gems/passenger-
 ${passenger_version}/bin/passenger-install-nginx-module
--auto --auto-download --prefix=/opt/nginx",
       creates =&gt; "/opt/nginx/sbin/nginx",
       require =&gt; Exec["install-passenger"],
       timeout =&gt; "-1",
   }

   file { [ "/opt/nginx",
            "/opt/nginx/conf",
            "/opt/nginx/conf/includes",
            "/opt/nginx/sites-enabled",
            "/opt/nginx/sites-available",
            "/var/log/nginx" ]:
       ensure =&gt; directory,
       owner =&gt; "www-data",
       group =&gt; "www-data",
   }

   file { "/opt/nginx/sites-enabled/default":
       ensure =&gt; absent,
       require =&gt; Exec["install-passenger-nginx-module"],
   }

   file { "/opt/nginx/conf/nginx.conf":
       content =&gt; template("rails/nginx.conf.erb"),
       notify  =&gt; Exec["reload-nginx"],
       require =&gt; Exec["install-passenger-nginx-module"],
   }

   file { "/etc/init.d/nginx":
       source  =&gt; "puppet:///modules/rails/nginx.init",
       mode    =&gt; "700",
       require =&gt; Exec["install-passenger-nginx-module"],
   }

   service { "nginx":
```

```
            enable =&gt; true,
            ensure =&gt; running,
            require =&gt; File["/etc/init.d/nginx"],
        }

        exec { "reload-nginx":
            command     =&gt; "/opt/nginx/sbin/nginx -t && /etc/ini
            refreshonly =&gt; true,
            require     =&gt; Exec["install-passenger-nginx-module"
        }
}
```

4. 使用如下内容创建 /etc/puppet/modules/rails/templates/app.conf.erb 文件：

```
server {
    listen 80;
    root /var/www/&lt;%= name %&gt;/current/public;
    server_name &lt;%= sitedomain %&gt;;
    access_log /var/log/nginx/&lt;%= name %&gt;.access.log;
    error_log /var/log/nginx/&lt;%= name %&gt;.error.log;

    passenger_enabled on;
    passenger_min_instances 1;
}

passenger_pre_start http://&lt;%= sitedomain %&gt;;
```

5. 使用如下内容创建 /etc/puppet/modules/rails/templates/nginx.conf.erb 文件：

```
events {
  worker_connections 1024;
  use epoll;
}

http {
  passenger_root /usr/lib/ruby/gems/1.8/gems/passenger-&lt;%=
  passenger_version %&gt;;

  server_names_hash_bucket_size 64;

  sendfile on;
  tcp_nopush on;
  tcp_nodelay off;
  client_body_temp_path /var/spool/nginx-client-body 1 2;

  client_max_body_size 100m;

  include /opt/nginx/conf/mime.types;
  default_type application/octet-stream;

  log_format main '$remote_addr - $remote_user [$time_local] '
          $request" $status $body_bytes_sent "$http_referer" '
          '"$http_user_agent" "$http_x_forwarded_for"' ;

  access_log /var/log/nginx/access.log main;

  gzip on;
  gzip_http_version 1.0;
  gzip_comp_level 2;
  gzip_proxied any;
  gzip_min_length 1100;
  gzip_buffers 16 8k;
  gzip_types text/plain text/html text/css application/x-javasc
   text/xml application/xml application/xml+rss text/javascript
  gzip_disable "MSIE [1-6].(?!.*SV1)";
  gzip_vary on;

  include /opt/nginx/sites-enabled/*;
}
```

6. 使用如下内容创建 /etc/puppet/modules/rails/files/nginx.init 文件：

```
#!/bin/sh

### BEGIN INIT INFO
# Provides: nginx
# Required-Start: $all
# Required-Stop: $all
# Default-Start: 2 3 4 5
```

```
# Default-Stop: 0 1 6
# Short-Description: starts the nginx web server
# Description: starts nginx using start-stop-daemon
### END INIT INFO

PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/b
DAEMON=/opt/nginx/sbin/nginx
NAME=nginx
DESC=nginx

test -x $DAEMON || exit 0

# Include nginx defaults if available
if [ -f /etc/default/nginx ] ; then
   . /etc/default/nginx
fi

set -e

# Return LSB status, grabbed from a newer lsb-base
status_of_proc () {
    local pidfile daemon name status

    pidfile=
    OPTIND=1
    while getopts p: opt ; do
 case "$opt" in
            p) pidfile="$OPTARG";;
        esac
 done

  shift $(($OPTIND - 1))

    if [ -n "$pidfile" ]; then
  pidfile="-p $pidfile"
    fi

  daemon="$1"
    name="$2"

    status="0"
    pidofproc $pidfile $daemon &gt;/dev/null || status="$?"
    if [ "$status" = 0 ]; then
  log_success_msg "$name is running"
        return 0
    else
  log_failure_msg "$name is not running"
        return $status
    fi
}

. /lib/lsb/init-functions
```

```
case "$1" in
  start)
    echo -n "Starting $DESC: "
    start-stop-daemon --start --quiet --pidfile /var/run/$NAME.
     --exec $DAEMON -- $DAEMON_OPTS || true
 echo "$NAME."
    ;;
  stop)
    echo -n "Stopping $DESC: "
    start-stop-daemon --stop --quiet --pidfile /var/run/$NAME.p
     --exec $DAEMON || true
 echo "$NAME."
    ;;
  restart|force-reload)
    echo -n "Restarting $DESC: "
    start-stop-daemon --stop --quiet --pidfile \
     /var/run/$NAME.pid --exec $DAEMON || true
  sleep 1
    start-stop-daemon --start --quiet --pidfile \
     /var/run/$NAME.pid --exec $DAEMON -- $DAEMON_OPTS || true
 echo "$NAME."
    ;;
  reload)
    echo -n "Reloading $DESC configuration: "
    start-stop-daemon --stop --signal HUP --quiet --pidfile \
     /var/run/$NAME.pid --exec $DAEMON || true
 echo "$NAME."
    ;;
  status)
    status_of_proc -p /var/run/$NAME.pid "$DAEMON" nginx \
     && exit 0 || exit $?
    ;;
  *)
    N=/etc/init.d/$NAME
    echo "Usage: $N {start|stop|restart|reload|forcereload|
 status}" &gt;&2
    exit 1
    ;;
esac
exit 0
```

7. 添加如下代码到一个节点：

```
rails::app { "furiouspigs":
    sitedomain =&gt; "furiouspigs.com",
}
```

8. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1309960678'

notice: /Stage[main]/Rails::Passenger/File[/opt/nginx]/ensure:
created

notice: /Stage[main]/Rails::Passenger/File[/opt/nginx/sitesenab
ensure: created

notice: /Stage[main]//Node[cookbook]/Rails::App[furiouspigs]/
File[/opt/nginx/sites-enabled/furiouspigs.conf]/ensure: createc

notice: /Stage[main]/Rails::Passenger/File[/opt/nginx/conf]/
ensure: created

notice: /Stage[main]/Rails::Passenger/File[/opt/nginx/conf/
includes]/ensure: created

notice: /Stage[main]//Node[cookbook]/Rails::App[furiouspigs]/
File[/opt/nginx/conf/includes/furiouspigs.conf]/ensure: definec
content as '{md5}d41d8cd98f00b204e9800998ecf8427e'

notice: /Stage[main]/Rails::Passenger/File[/opt/nginx/sitesavai
ensure: created

notice: /Stage[main]//Node[cookbook]/Rails::App[furiouspigs]/
File[/opt/nginx/sites-available/furiouspigs.conf]/ensure: defir
content as '{md5}c1a4c2bc4e7381b1c2f88dfee004a594'

notice: /Stage[main]/Rails::Passenger/Exec[install-passenger]/
returns: executed successfully

notice: /Stage[main]/Rails::Passenger/Exec[install-passengerngi
module]/returns: executed successfully
--- /opt/nginx/conf/nginx.conf 2011-07-06 14:04:33.231999538
+0000
+++ /tmp/puppet-file20110706-5343-k8ouds-0 2011-07-06
14:04:34.246867124 +0000
...

info: /Stage[main]/Rails::Passenger/File[/opt/nginx/conf/nginx.
conf]: Filebucketed /opt/nginx/conf/nginx.conf to puppet with s
34d60856b6570e9d59cd6eecde5da000

notice: /Stage[main]/Rails::Passenger/File[/opt/nginx/conf/ngir
conf]/content: content changed '{md5}34d60856b6570e9d59cd6eecde
da000' to '{md5}72132deeb45e6ee5b83cd246dffefc5f'
info: /Stage[main]/Rails::Passenger/File[/opt/nginx/conf/nginx.
conf]: Scheduling refresh of Exec[reload-nginx]
notice: /Stage[main]/Rails::Passenger/Exec[reload-nginx]:
```

```
Triggered 'refresh' from 1 events
notice: Finished catalog run in 398.73 seconds
```

工作原理

这个处方比本书中其他的处方更长、更复杂，因此需要更详细的解释。 如果你觉得这有些烦人，尽管去使用这个配方吧，不必担心它是如何工作的。 稍后当你想学习更多详细工作过程时，可以再回过头来看这些解释。

上面所有代码的目的就是可以让你写出如下的实例化代码：

```
rails::app { "furiouspigs":
    sitedomain => "furiouspigs.com",
}
```

这需要不少幕后工作。我们需要安装配置包含 Passenger 模块的 Nginx， 配置应用程序的虚拟主机，包括所有应用程序特定的配置（比如重定向以及其他服务配置）， 安装 Ruby 和 Rubygems，安装 Bundler，并创建要部署的应用程序所需的所有目录。

Nginx 和 Passenger

此处分离出的 passenger.pp 文件用于安装 Nginx 和 Passenger 所需的一切。 之前曾经提到过，Nginx 没有像 Apache 一样的动态模块概念， 因此，你不能仅仅通过安装发行版中的 Nginx 并安装提供 Passenger 功能的软件包来实现。 Nginx 必须与你想要的任何模块一起编译。

幸好，**Phusion** 社区里的好心人已经为我们提供了一个编译脚本（passengerinstall-nginx-module）。 一旦你已经安装了 Passenger 的 gem 包，这个脚本就会帮你完成编译工作。 所以首先需要做的事就是安装 Passenger 的 **gem**：

```
class rails::passenger {
    $passenger_version = "3.0.7"
    $passenger_dependencies = [ "build-essential",
                                "libcurl4-openssl-dev",
                                "libssl-dev",
                                "ruby",
                                "rubygems" ]
    package { $passenger_dependencies: ensure => installed }

    exec { "install-passenger":
        command => "/usr/bin/gem install passenger
  --version=${passenger_version}",
        unless => "/usr/bin/gem list | /bin/grep passenger \
  |/bin/grep ${passenger_version}",
        require => [ Package["rubygems"], Package[$passenger_
         dependencies] ],
        timeout => "-1",
    }
```

我们把要安装的 Passenger 版本号设置在变量 $passenger_version 中， 因为 Nginx 需要知道 Passenger 的安装路径（路径中包括版本号）。 所以我们会在 nginx.conf 模板中引用 $passenger_version 变量。

下一步是运行 passenger-install-nginx-module 脚本：

```
    exec { "install-passenger-nginx-module":
        command => "/usr/lib/ruby/gems/1.8/gems/passenger-${passeng
  version}/bin/passenger-install-nginx-module --auto --autodownload
  --prefix=/opt/nginx",

        creates => "/opt/nginx/sbin/nginx",
        require => Exec["install-passenger"],
        timeout => "-1",
    }
```

> 你应该注意到了，此处 **gem** 的路径是固定的 /usr/lib/ruby/gems/1.8/gems。 这有些脆弱?—?在大部分生产基础设施中，我使用 **RVM** 管理 Ruby 版本和不同版本的 **gemsets**， 这就解决了由于使用固定路径带来的脆弱性。 然而，添加 RVM 会使本处方变得难于理解，所以我将使用 RVM 的部分放在 更多用法 小节中。 一旦你熟悉了这个处方，就可以修改它，使之适合你自己的需求，包括整合 RVM。

这也就意味着，如果你使用 Ruby 1.9，这个处方不会工作，当本书出版后你读到此处时很可能会发生这种情况。 如果是这样，或者你遇到其他问题，请手工运行 gem contents passenger 命令之后查找 passenger-installnginx-module 脚本的路

径。

下一步，我们创建了 Nginx 配置文件所需的目录结构：

```
file { [ "/opt/nginx",
         "/opt/nginx/conf",
         "/opt/nginx/conf/includes",
         "/opt/nginx/sites-enabled",
         "/opt/nginx/sites-available",
         "/var/log/nginx" ]:
    ensure => directory,
    owner  => "www-data",
    group  => "www-data",
}
```

我们要删除默认的 Nginx 虚拟主机配置，否则可能会干扰我们要创建的虚拟主机。这可以通过如下代码实现：

```
file { "/opt/nginx/sites-enabled/default":
    ensure  => absent,
    require => Exec["install-passenger-nginx-module"],
}
```

实际上，如果你从源代码构建 Nginx 或是通过 Passenger 构建（本例中的方法），这是不需要的， 但是如果要想使本处方也适用于发行版的 Nginx 软件包，这会是有益的。

下面是 Nginx 的主配置文件：

```
file { "/opt/nginx/conf/nginx.conf":
    content => template("rails/nginx.conf.erb"),
    notify  => Exec["reload-nginx"],
    require => Exec["install-passenger-nginx-module"],
}
```

主配置文件使用 nginx.conf.erb 模板生成，因为我们需要插入之前定义的 Passenger 版本号：

```
passenger_root /usr/lib/ruby/gems/1.8/gems/passenger-<%= passenger_
version %>;
```

否则，它就是一个标准的 Nginx 配置，你也可以在模板中添加任何你的服务器所需的特殊参数。

因为我们没有使用发行版提供的软件包，所以我们需要为节点应用一个 init 脚本（改编自 Ubuntu 版本，仅做了轻微的修改）：

```
file { "/etc/init.d/nginx":
    source  => "puppet:///modules/rails/nginx.init",
    mode    => "700",
    require => Exec["install-passenger-nginx-module"],
}
```

为了运行 Nginx 服务需要如下代码：

```
service { "nginx":
    enable  => true,
    ensure  => running,
    require => File["/etc/init.d/nginx"],
}
```

要确保错误的配置不会使服务器宕机，配置文件的改变会通知如下的“配置检查和重载”资源：

```
exec { "reload-nginx":
    command     => "/opt/nginx/sbin/nginx -t \
 && /etc/init.d/nginx reload",
    refreshonly => true,
    require     => Exec["install-passenger-nginx-module"],
}
```

Rails

你已经设置好了 Passenger 和 Nginx，接下来配置 **Rails** 需求的类：

```
class rails {
    include rails::passenger

    package { "bundler":
        provider => gem,
        ensure => installed,
    }
}
```

Bundler 是一个管理应用程序或解决 gem 依赖关系的工具。 你可以使用手工方式（或通过 Puppet）指定要安装的所有 gem 包以及依赖的包， 取代这种方式的更好办法是使用 Bundler 来实现，它是 Rails 部署的一部分。 例如，你应该注意到了，我们没有安装 rails 的 gem；这通常是通过 Bundler 安装的， 或者在应用程序的

vendor 目录中已经提供了一份特定版本的 rails。 如果你没有使用 Bundler，或者你需要为你的 Rails 应用程序设置一些额外的依赖， 请在这个类中使用 Puppet 的 package 资源做相应的配置来安装它们。

rails 类的的主要部分是 define 函数 app， 对于你要管理的每个应用程序，它都会被实例化一次：

```
define app( $sitedomain ) {
    include rails
```

为应用程序做的第一件事是安装 Nginx 虚拟主机配置文件，它由 app.conf.erb 模板生成：

```
    file { "/opt/nginx/sites-available/${name}.conf":
        content => template("rails/app.conf.erb"),
        require => File["/opt/nginx/sites-available"],
    }

    file { "/opt/nginx/sites-enabled/${name}.conf":
        ensure => link,
        target => "/opt/nginx/sites-available/${name}.conf",
        require => File["/opt/nginx/sites-enabled"],
        notify => Exec["reload-nginx"],
    }
```

虚拟主机配置文件的模板相当小：

```
server {
    listen 80;
    root /var/www/<%= name %>/current/public;
    server_name <%= sitedomain %>;
    access_log /var/log/nginx/<%= name %>.access.log;
    error_log /var/log/nginx/<%= name %>.error.log;

    passenger_enabled on;
    passenger_min_instances 1;
}

passenger_pre_start http://<%= sitedomain %>;
```

通常一个应用程序需要特定的 Nginx 配置指令，比如 redirects。 你可以在 Rails 模块中添加一个名为 files/furiouspigs.conf 的文件来包含这些配置指令。 Puppet 会从如下的这段代码找到这个文件，并分发给节点：

```
    file { "/opt/nginx/conf/includes/${name}.conf":
        source => [ "puppet:///modules/rails/${name}.conf",
                    "puppet:///modules/rails/empty.conf" ],
        notify => Exec["reload-nginx"],
    }
```

注意这个文件使用了多个源，第二个源 empty.conf 确保 Puppet 不会因为应用程序指定的配置文件不存在而抱怨。

最后我们要确保为准备部署的标准 Rails 目录结构配置 www-data 用户属主及适当的权限（775）。如果你的 Nginx 以及部署的应用程序以不同的用户身份执行，请使用你的用户名替换所有的 www-data。

```
       file { [ "/var/www",
                "/var/www/${name}",
                "/var/www/${name}/releases",
                "/var/www/${name}/shared",
                "/var/www/${name}/shared/config",
                "/var/www/${name}/shared/log",
                "/var/www/${name}/shared/system" ]:
            ensure => directory,
            mode   => 775,
            owner  => "www-data",
            group  => "www-data",
        }
    }
}
```

更多用法

使用 Puppet 管理 Rails 应用程序，你可能还需要考虑一些其他的事情。

RVM

正如我之前所提到的，使用 RVM 管理多版本 Ruby 和多版本的 gemsets 是一种强大的解决方案。当然，使用 RVM 也带来了自身的有趣问题?—?RVM 的开发活跃，其主体经常变化。无论如何，RVM 为我们带来的好处还是比其带来的麻烦要多的多。所以，我建议你对生产线上的 Rails 站点，或许可以通过类似这样的代码应用 RVM：

```
class rails::rvm {
  package { [ "autoconf",
              "bison",
              "curl",
              "libreadline-dev",
              "subversion",
              "zlib1g-dev" ]:
      ensure => installed
  }

  file { "/usr/local/bin/rvm-install-system-wide":
      source => "puppet:///modules/rails/rvm-install-system-wide",
      mode   => "700",
  }

  exec { "install-rvm":
      command => "/usr/local/bin/rvm-install-system-wide",
      creates => "/usr/local/bin/rvm",
      require => [ Package["curl"], Package["subversion"],
                   File["/usr/local/bin/rvm-install-system-wide"] ]
      logoutput => on_failure,
  }

  append_if_no_such_line { "setup-rvm-shell-environment":
      file => "/etc/bash.bashrc",
      line => "[[ -s /usr/local/rvm/scripts/rvm ]] \
 && . /usr/local/rvm/scripts/rvm",
  }
}
```

rvm-install-system-wide 脚本来自 RVM 网站：https://rvm.beginrescueend.com/install/rvm 。

日志滚动

在生产环境中你可能会需要为 Nginx 和 Rails 生成的日志文件添加 logrotate 配置片段，以确保这些日志不会逐步占满你的磁盘。因为篇幅限制和保持简单的原因，本例省略了对日志滚动的配置。

数据库

本处方中，未对 Rails 应用程序创建任何数据库及访问数据库的用户；配置何种数据库取决于你团队的开发者正在使用何种数据库（MySQL、Postgres、MongoDB 等），你需要自行添加管理数据库的 Puppet 代码。如果使用的是 MySQL，你可以参考 创建 MySQL 数据库及用户 一节的内容并作适当的改写。

SSL 证书

一些应用程序会需要 SSL 证书并为 **vhost** 配置安全的 URLs，例如：处理网上支付。 这已超出了本处方的讲解范围，但你应该可以发现添加所需的代码并不困难。你可以在 rails::app 函数的定义中添加一个可选参数，例如：

```
define app( $sitedomain, $ssl = false ) {
```

并添加如下代码处理这个参数：

```
if $ssl {
   file { "/etc/ssl/certs/${name}.crt":
       source => "puppet:///modules/rails/${name}.crt",
   }
}
```

然后，只要使用如下的代码对你的应用程序进行实例化即可：

```
rails::app { "irritatedbadgers":
    sitedomain => "irritatedbadgers.com",
    ssl        => true,
}
```

服务器和云基础设施

> Rest is not idleness, and to lie sometimes on the grass under trees on a summer's day, listening to the murmur of the water, or watching the clouds float across the sky, is by no means a waste of time.
>
> — J. Lubbock

在本章中，我们将学习如下内容：

- 部署 Nagios 监控服务器
- 使用 Heartbeat 构建高可用服务
- 管理 NFS 服务和文件共享
- 使用 HAProxy 为多个 web 服务器实现负载均衡
- 使用 iptables 管理防火墙
- 管理 Amazon 的 EC2 实例
- 使用 Vagrant 管理虚拟机

Puppet 的强大不仅在于它可以管理配置单台服务器还可以控制网络中的大量服务器， 只有控制大量服务器时才能真正显现其效益。 在本章中我们将探讨：如何使用 Puppet 帮助你监控基础设施， 创建高可用性集群，跨越网络分享文件，设立自动的防火墙， 为你的众多后端服务器实现负载均衡，以及在云中或桌面上创建新虚拟机。

部署 Nagios 监控服务器

> My roommate lost his pet elephant. It's in the apartment somewhere.
>
> — Steven Wright

我们无法持续关注所有的一切。问题是：你怎么知道一台服务器何时会出现故障呢？ 错误的答案是，“我的客户打电话给我，告诉我服务器宕机了。” 但是，你会惊奇地发现，许多组织的系统没有任何的自动监控。自动监控的设置非常简单。 目前有许多优秀的免费开源的自动监测工具可用，包括 Nagios、Icinga、Zabbix 和 Zenoss。 Nagios 是其中历史最久最复杂的一个，尽管它有一个难以配置的名声（基本上是实至名归的）。

本处方将向你展示如何使用 Puppet 创建一个基于 Nagios 的监控服务器以及 如何让 Puppet 配置由 Nagios 监控的每台机器。

准备工作

你将需要我们已经在第 7 章的 管理 Apache 服务 一节中创建的 Apache 模块。

操作步骤

1. 创建一个 nagios 模块：

```
# mkdir /etc/puppet/modules/nagios
# mkdir /etc/puppet/modules/nagios/files
# mkdir /etc/puppet/modules/nagios/manifests
```

2. 使用如下内容创建 /etc/puppet/modules/nagios/manifests/server.pp 文件：

```
class nagios::server {
    include apache

    package { [ "nagios3",
                "nagios-images",
                "nagios-nrpe-plugin" ]:
        ensure =&gt; installed,
    }

    service { "nagios3":
        ensure =&gt; running,
        enable =&gt; true,
        require =&gt; Package["nagios3"],
    }

    exec { "nagios-config-check":
        command =&gt; "/usr/sbin/nagios3 -v /etc/nagios3/nagios
```

```
cfg && /usr/sbin/service nagios3 restart",
        refreshonly =&gt; true,
    }

    file { "/etc/apache2/sites-available/nagios.conf":
        source  =&gt; "puppet:///modules/nagios/nagios.conf",
        notify  =&gt; Service["apache2"],
        require =&gt; Package["apache2-mpm-prefork"],
    }

    file { "/etc/apache2/sites-enabled/nagios.conf":
        ensure  =&gt; symlink,
        target  =&gt; "/etc/apache2/sites-available/nagios.conf
        require =&gt; Package["apache2-mpm-prefork"],
    }

    file { [ "/etc/nagios3/generic-service_nagios2.cfg",
             "/etc/nagios3/services_nagios2.cfg",
             "/etc/nagios3/hostgroups_nagios2.cfg",
             "/etc/nagios3/extinfo_nagios2.cfg",
             "/etc/nagios3/localhost_nagios2.cfg",
             "/etc/nagios3/contacts_nagios2.cfg",
             "/etc/nagios3/conf.d"
           ]:
        ensure =&gt; absent,
        force  =&gt; true,
    }

    define nagios-config() {
        file { "/etc/nagios3/${name}":
            source  =&gt; "puppet:///modules/nagios/${name}",
            require =&gt; Package["nagios3"],
            notify =&gt; Exec["nagios-config-check"],
        }
    }

    nagios-config { [ "htpasswd.nagios",
                      "nagios.cfg",
                      "cgi.cfg",
                      "hostgroups.cfg",
                      "hosts.cfg",
                      "host_templates.cfg",
                      "service_templates.cfg",
                      "services.cfg",
                      "timeperiods.cfg",
                      "contacts.cfg",
                      "commands.cfg" ]: }

    file { "/var/lib/nagios3":
        # see http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=
        mode    =&gt; 751,
        require =&gt; Package["nagios3"],
        notify  =&gt; Service["nagios3"],
```

```
    }

    file { "/var/lib/nagios3/rw":
        # see http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=
        mode    =&gt; 2710,
        require =&gt; Package["nagios3"],
        notify  =&gt; Service["nagios3"],
    }
}
```

3. 使用如下内容创建 /etc/puppet/modules/nagios/files/nagios.cfg 文件：

```
# Config files to read
cfg_file=/etc/nagios3/commands.cfg
cfg_file=/etc/nagios3/service_templates.cfg
cfg_file=/etc/nagios3/host_templates.cfg
cfg_file=/etc/nagios3/timeperiods.cfg
cfg_file=/etc/nagios3/contacts.cfg
cfg_file=/etc/nagios3/hostgroups.cfg
cfg_file=/etc/nagios3/hosts.cfg
cfg_file=/etc/nagios3/services.cfg

# Nagios settings
log_file=/var/log/nagios3/nagios.log
illegal_macro_output_chars=`~$&|'"&lt;&gt;
check_result_path=/var/lib/nagios3/spool/checkresults
nagios_user=nagios
nagios_group=nagios
command_file=/var/lib/nagios3/rw/nagios.cmd
lock_file=/var/run/nagios3/nagios3.pid
p1_file=/usr/lib/nagios3/p1.pl
check_external_commands=1
resource_file=/etc/nagios3/resource.cfg
```

4. 使用如下内容创建 /etc/puppet/modules/nagios/files/service_templates.cfg 文件：

```
define service{
    name                          generic_service
     ; The 'name' of this service template
    active_checks_enabled         1
     ; Active service checks are enabled
    passive_checks_enabled        1
     ; Passive service checks are enabled/accepted
    parallelize_check             1
     ; Active service checks should be parallelized
     ; (disabling this can lead to major performance problems)
    obsess_over_service           1
     ; We should obsess over this service (if necessary)
```

```
    check_freshness               0
     ; Default is to NOT check service 'freshness'
    notifications_enabled         1
     ; Service notifications are enabled
    event_handler_enabled         1
     ; Service event handler is enabled
    flap_detection_enabled        1
     ; Flap detection is enabled
    failure_prediction_enabled    1
     ; Failure prediction is enabled
    process_perf_data             1
     ; Process performance data
    retain_status_information     1
     ; Retain status information across program restarts
    retain_nonstatus_information 1
     ; Retain non-status information across program restarts
    notification_interval         0
     ; Only send notifications on status change by default.
    is_volatile                   0
    check_period                  24x7
    normal_check_interval         5
    retry_check_interval          2
    max_check_attempts            3
    notification_period           24x7
    notification_options          c,r
    contact_groups                sysadmin
    register                      0
     ; DONT REGISTER THIS DEFINITION
     ;   - ITS NOT A REAL SERVICE, JUST A TEMPLATE!
}

# Defaults
define service {
    name                          every_5_mins
    normal_check_interval         5
    use                           generic_service
    register                      0
}

define service {
    name                          every_hour
    normal_check_interval         60
    use                           generic_service
    register                      0
}

define service {
    name                          every_day
    normal_check_interval         1440
    use                           generic_service
    register                      0
}
```

5. 使用如下内容创建 /etc/puppet/modules/nagios/files/services.cfg 文件：

```
define service {
    hostgroup_name                  all
    service_description             Disk
    check_command                   check_nrpe!check_all_disks!20%
    use                             every_day
}

define service {
    hostgroup_name                  all
    service_description             Load
    check_command                   check_nrpe!check_load!10,10,10
    use                             every_hour
}
```

6. 使用如下内容创建 /etc/puppet/modules/nagios/files/cgi.cfg 文件：

```
main_config_file=/etc/nagios3/nagios.cfg
physical_html_path=/usr/share/nagios3/htdocs
url_html_path=/nagios3
show_context_help=1
use_pending_states=1
nagios_check_command=/usr/lib/nagios/plugins/check_nagios
 /var/cache/nagios3/status.dat 5 '/usr/sbin/nagios3'
use_authentication=1
use_ssl_authentication=0
authorized_for_system_information=nagios
authorized_for_configuration_information=nagios
authorized_for_system_commands=nagios
authorized_for_all_services=nagios
authorized_for_all_hosts=nagios
authorized_for_all_service_commands=nagios
authorized_for_all_host_commands=nagios
default_statusmap_layout=5
default_statuswrl_layout=4
ping_syntax=/bin/ping -n -U -c 5 $HOSTADDRESS$
refresh_rate=90
escape_html_tags=1
action_url_target=_blank
notes_url_target=_blank
lock_author_names=1
```

7. 使用如下内容创建 /etc/puppet/modules/nagios/files/host_templates.cfg 文件：

```
define host{
    name                           generic_host
    check_command                  check-host-alive
    max_check_attempts             3
    checks_enabled                 1
    failure_prediction_enabled     1
    retain_status_information      1
    retain_nonstatus_information   1
    notification_interval          0
    notification_options           d,u,r
    check_interval                 300
    contact_groups                 sysadmin
    register                       0
}
```

8. 使用如下内容创建 /etc/puppet/modules/nagios/files/contacts.cfg 文件（使用你自己的 e-mail 地址，或者至少是一个不介意从你的监控服务获得大量邮件的人的 e-mail 地址）：

```
define contact {
    contact_name                     helen
    alias Helen                      Highwater
    service_notification_period      24x7
    host_notification_period         24x7
    service_notification_options     w,u,c,r
    host_notification_options        d,r
    service_notification_commands    notify-service-by-email
    host_notification_commands       notify-host-by-email
    email                            helen@example.com
}

define contactgroup {
    contactgroup_name                sysadmin
    alias                            Sysadmins
    members                          helen
}
```

9. 使用如下内容创建 /etc/puppet/modules/nagios/files/hostgroups.cfg 文件：

```
define hostgroup {
    hostgroup_name                    all
    alias                             All Servers
    members                           *
}
```

10. 使用如下内容创建 /etc/puppet/modules/nagios/files/timeperiods.cfg 文件：

```
define timeperiod {
   timeperiod_name      24x7
   alias                24 Hours A Day, 7 Days A Week
   sunday               00:00-24:00
   monday               00:00-24:00
   tuesday              00:00-24:00
   wednesday            00:00-24:00
   thursday             00:00-24:00
   friday               00:00-24:00
   saturday             00:00-24:00
}
```

11. 使用如下内容创建 /etc/puppet/modules/nagios/files/hosts.cfg 文件（使用你自己的服务器信息替换相应的配置值）：

```
define host {
   host_name           cookbook
   address             cookbook.bitfieldconsulting.com
   use                 generic_host
}
```

12. 使用如下内容创建 /etc/puppet/modules/nagios/files/commands.cfg 文件：

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
    -a $ARG2$ $ARG3$ $ARG4$ $ARG5$
}

define command{
   command_name  check-host-alive
   command_line  $USER1$/check_ping -H '$HOSTADDRESS$' -w
    5000,100% -c 5000,100% -p 1
}

define command{
   command_name  check_all_disks
   command_line  /usr/lib/nagios/plugins/check_disk -w '$ARG1$'
    -c '$ARG2$' -e
}

define command{
   command_name   notify-host-by-email
   command_line   /usr/bin/printf "%b" "***** Nagios *****\n\
    nNotification Type: $NOTIFICATIONTYPE$\nHost: $HOSTNAME$\
    nState: $HOSTSTATE$\nAddress: $HOSTADDRESS$\nInfo:
    $HOSTOUTPUT$\n\nDate/Time: $LONGDATETIME$\n" | /usr/bin/mai
    -s "** $NOTIFICATIONTYPE$ Host Alert: $HOSTNAME$ is
    $HOSTSTATE$ **" $CONTACTEMAIL$
}

define command{
   command_name  notify-service-by-email
   command_line  /usr/bin/printf "%b" "***** Nagios
    *****\n\nNotification Type: $NOTIFICATIONTYPE$\n\nService:
    $SERVICEDESC$\nHost: $HOSTALIAS$\nAddress: $HOSTADDRESS$\
    nState: $SERVICESTATE$\n\nDate/Time: $LONGDATETIME$\n\
    nAdditional Info:\n\n$SERVICEOUTPUT$" | /usr/bin/mail -s "*
    $NOTIFICATIONTYPE$ Service Alert: $HOSTALIAS$/$SERVICEDESC$
    $SERVICESTATE$ **" $CONTACTEMAIL$
}
```

13. 使用如下内容创建 /etc/puppet/modules/nagios/files/nagios.conf 文件（用你自己的服务器替换 ServerName 的配置值）：

```
ScriptAlias /cgi-bin/nagios3 /usr/lib/cgi-bin/nagios3
ScriptAlias /nagios3/cgi-bin /usr/lib/cgi-bin/nagios3

Alias /nagios3/stylesheets /etc/nagios3/stylesheets
Alias /nagios3 /usr/share/nagios3/htdocs
Alias / /usr/share/nagios3/htdocs/

&lt;DirectoryMatch (/usr/share/nagios3/htdocs|/usr/lib/cgi-bin/
nagios3|/etc/nagios3/stylesheets)&gt;

   Options FollowSymLinks

   DirectoryIndex index.html

   AllowOverride AuthConfig

   Order Allow,Deny
   Allow From All

   AuthName "Nagios Access"
   AuthType Basic
   AuthUserFile /etc/nagios3/htpasswd.nagios
   require valid-user
&lt;/DirectoryMatch&gt;

&lt;VirtualHost *:80&gt;
   ServerName nagios.bitfieldconsulting.com
   ErrorLog /var/log/apache2/nagios-error_log
   CustomLog /var/log/apache2/nagios-access_log common
   DocumentRoot /usr/share/nagios3
&lt;/VirtualHost&gt;
```

14. 创建口令文件控制对 Nagios web 界面的访问：

```
# htpasswd -c /etc/puppet/modules/nagios/files/htpasswd.nagios
nagios
Password: (type password)
```

15. 如果你的系统中还没有 htpasswd 程序，运行如下的命令：

```
# apt-get install apache2-utils
```

16. 在前面的代码中指定的 ServerName 为你创建一个 /etc/hosts 条目或 DNS 记录，本例中的主机名是：

```
nagios.bitfieldconsulting.com
```

17. 在你的 Nagios 服务器的节点中定义中包含如下代码：

```
include nagios::server
```

18. 使用如下内容创建 /etc/puppet/modules/nagios/files/nrpe.cfg 文件（使用你自己的监控服务器的主机名或 IP 地址替换 allowed_hosts 设置值）：

```
log_facility=daemon
pid_file=/var/run/nagios/nrpe.pid
server_port=5666
nrpe_user=nagios
nrpe_group=nagios
allowed_hosts=cookbook.bitfieldconsulting.com
dont_blame_nrpe=1
debug=0
command_timeout=60
connection_timeout=300
command[check_load]=/usr/lib/nagios/plugins/check_load -w $ARG1
-c $ARG2$
command[check_all_disks]=/usr/lib/nagios/plugins/check_disk -w
$ARG1$ -c $ARG2$ -e -A -i '.gvfs'
```

19. 使用如下内容创建 /etc/puppet/modules/nagios/manifests/target.pp 文件：

```
class nagios::target {
   package { [ "nagios-nrpe-server",
               "nagios-plugins",
               "nagios-plugins-basic",
               "nagios-plugins-standard",
               "nagios-plugins-extra" ]:
       ensure =&gt; installed,
   }

   service { "nagios-nrpe-server":
       enable  =&gt; true,
       ensure  =&gt; running,
       pattern =&gt; "/usr/sbin/nrpe",
       require =&gt; Package["nagios-nrpe-server"],
   }

   file { "/etc/nagios/nrpe.cfg":
       source  =&gt; "puppet:///modules/nagios/nrpe.cfg",
       require =&gt; Package["nagios-nrpe-server"],
       notify  =&gt; Service["nagios-nrpe-server"],
   }
}
```

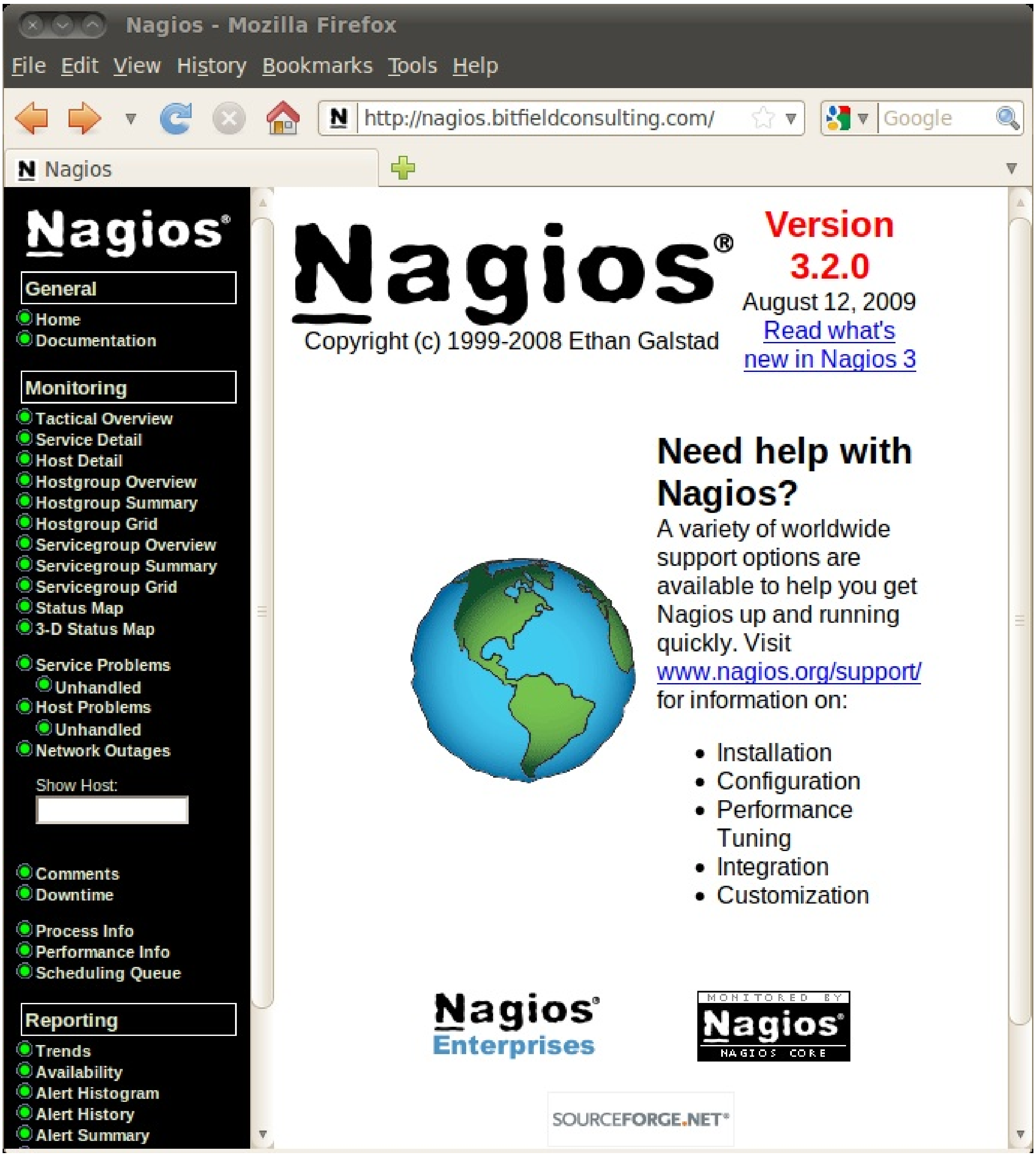
20. 在你要监控的所有节点（也应该包括 Nagios 服务器自己）上包含这个类：

```
include nagios::target
```

21. 在 Nagios 服务器上运行如下 Puppet 命令：

```
# puppet agent --test
```

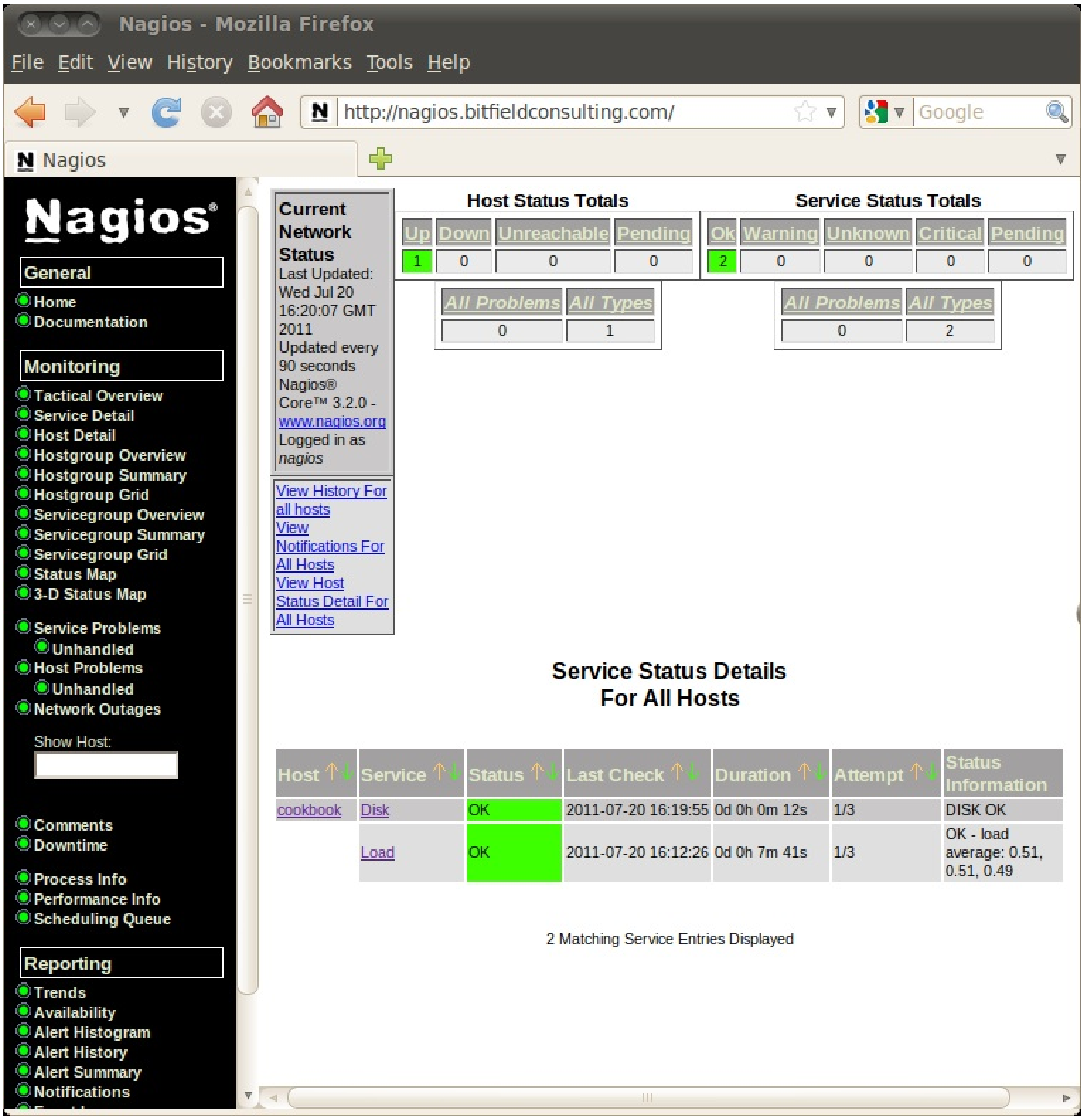
22. 使用浏览器打开 Nagios 的 web 界面（使用用户名 nagios 和你在前面的代码中设置的口令登录） 检查你看到的 Nagios 欢迎界面，如图所示：



23. 点击 **Host detail** 菜单，在界面中你应该看到目标节点的列表。

24. 点击节点的名字，之后从 **Host commands** 菜单中选择 "Schedule a check of all services on this host"。

25. 在 "Force check" 复选框上打勾将其选中并单击 "Commit"。 这会花几秒钟运行 Nagios 的检查。 点击 "Service detail" 菜单，在界面中你应该看到显示为绿色的服务列表，如图所示：



工作原理

尽管这个处方相当的长，但是 Puppet 的配置清单本身却很简单； 这里没有我们在其他处方中还没使用过的配置技巧。

本质上，我们所做的就是安装 Nagios 包，配置服务于它的 Apache 虚拟主机， 为 Nagios 部署一系列的配置文件，这些配置文件用于告诉 Nagios 要检查哪些主机、检查哪些服务以及其他一些杂七杂八的配置。

在客户端，由 Nagios 监控的每个节点上都需要安装 nagios-nrpe-server 包（NRPE 是使 Nagios 能在远程服务器上安全地执行命令的协议） 以及告诉 NRPE 服务允许执行哪些命令的配置文件。

> 你应该知道，nrpe.cfg 文件中的 dont_blame_nrpe 设置是一个潜在的安全漏洞， 因为它允许在远程主机上使用用户提供的参数去执行命令。 这是一个非常有用的功能，因为这意味着你可以在无需重新配置每一个监控机的情况下，就可以改变警报阈值或其他参数。 然而，如果你不需要这个功能，禁用 dont_blame_nrpe 设置会更安全。

更多用法

使用 Nagios 最为棘手的事情，就是获取、配置及首次执行。 虽然这里介绍的是非常基本的监控配置（仅包括对磁盘占用和 CPU 负载的检查）， 你可以使用这个配置作为 Nagios 设置工作的起点，添加更多要检查的服务和主机。 你可能想要添加如下的一些配置：

- 主机组（例如，web 服务器组或数据库服务器组）：你可以配置一个检查自动应用到主机组的每个成员。
- Web 站点检查：配置 Nagios 的 check_http plugin 插件是相当复杂的， 它可以处理重定向、SSL、认证以及在一个 Web 页面中匹配文本。
- 进程检查：监视一个主机上指定的进程是一个常见的需求。使用 check_procs 插件可以实现。
- 不同的检查频率：我已经在 service_templates.cfg 模板中定义了 every_hour、 every_day 和 every_5_mins；你可能想要添加一些新的频率设置。
- 新的时间周期；在 timeperiods.cfg 中，当前仅定义了一个时间周期 24x7， 但你可能想要创建自己的时间周期。例如，如果要从午夜 0 点到凌晨 1 点执行数据库的维护工作， 你可以定义一个排除了这段时间的时间周期，而不会获得来自这个数据库服务器的误报。

要找到如何配置 Nagios 的更多内容, 请参考文档：http://nagios.sourceforge.net/docs/nagioscore/3/en/toc.html 。

在 Puppet 中也有一些对 Nagios 内置支持；你可以让 Puppet 从配置清单生成主机和服务的定义， 这是一个强大而有用的功能。尽管我在一些生产站点上会使用这个功能，但基于篇幅的原因， 我不得不遗憾地排除了对它的介绍。若你希望找到与这部分的相关的内容，请参考 Puppet 的官方文档以及 Mike Gurski 撰写的有关这个主题的优秀文章： http://blog.gurski.org/index.php/2010/01/28/automatic-monitoring-with-puppet-and-nagios/ 。

使用 Heartbeat 构建高可用服务

> Even in the future, nothing works!
>
> — Spaceballs

一切迟早都会发生故障。高可用服务就是指当一个主机或网络线路失效时仍旧能够提供服务。 高可用性的主要技术就是冗余，另外，这个问题的解决就是以投放更多硬件设备而著称的。

虽然最终肯定会有单独的一台服务器失效，但是两台服务器同时失效的概率是不太高的， 这对大多数的应用程序提供了一个良好的冗余水平。

最简单的方法之一是建立一对冗余服务器，它们共享一个 IP 地址，并使用心跳检测。 心跳（**Heartbeat**）是一个守护进程，它同时运行在两台机器上并且定期彼此交换信息（heartbeats）。 其中的一台是主服务器，通常它拥有资源：在本例中是一个 IP 地址。 如果辅助服务器无法从主服务器检测到心跳信号，它就接管地址以确保服务的连续性。

在下面的处方中，我们将使用 Puppet 的配置设置两台机器，并解释如何使用它提供一个高可用服务。

操作步骤

1. 创建如下的 heartbeat 模块：

   ```
   # mkdir /etc/puppet/modules/heartbeat
   # mkdir /etc/puppet/modules/heartbeat/manifests
   # mkdir /etc/puppet/modules/heartbeat/files
   ```

2. 使用如下内容创建 /etc/puppet/modules/heartbeat/manifests/init.pp 文件：

```
class heartbeat {
  package { "heartbeat":
    ensure =&gt; installed,
  }

  service { "heartbeat":
    ensure  =&gt; running,
    require =&gt; Package["heartbeat"],
  }

  exec { "reload-heartbeat":
    command     =&gt; "/usr/sbin/service heartbeat reload",
    refreshonly =&gt; true,
  }

  file { "/etc/ha.d/authkeys":
    source  =&gt; "puppet:///modules/heartbeat/authkeys",
    mode    =&gt; "600",
    require =&gt; Package["heartbeat"],
    notify  =&gt; Exec["reload-heartbeat"],
  }

  file { "/etc/ha.d/haresources":
    source  =&gt; "puppet:///modules/heartbeat/haresources",
    notify  =&gt; Exec["reload-heartbeat"],
    require =&gt; Package["heartbeat"],
  }

  file { "/etc/ha.d/ha.cf":
    source  =&gt; "puppet:///modules/heartbeat/ha.cf",
    notify  =&gt; Exec["reload-heartbeat"],
    require =&gt; Package["heartbeat"],
  }
}
```

3. 使用如下内容创建 /etc/puppet/modules/heartbeat/files/haresources 文件。将 cookbook 替换成你的主服务器的主机名。这可以通过在主服务器上运行 uname -n 命令获得。将 10.0.2.100 替换成你要在两台主机上共享的 IP 地址（这应该是当前网络上还未使用的地址）。最后列出的接口是分配给心跳检测的（本例中是 eth0:1）。

```
cookbook IPaddr::10.0.2.100/24/eth0:1
```

4. 使用如下内容创建 /etc/puppet/modules/heartbeat/files/authkeys 文件（使用你自己选择的口令替换 topsecretpassword）：

```
auth 1
1 sha1 topsecretpassword
```

5. 使用如下内容创建 /etc/puppet/modules/heartbeat/files/ha.cf 文件。 替换下面的两个 IP 地址为你自己的两台机器上 eth0 接口对应的 IP 地址。 同样需替换 cookbook 和 cookbook2 为你自己的两台机器的主机名（可以通过运行 uname -n 命令获得）。

```
autojoin none
ucast eth0 10.0.2.15
ucast eth0 10.0.2.16
keepalive 1
deadtime 10
warntime 5
udpport 694
auto_failback on
node cookbook
node cookbook2
use_logd yes
```

6. 在两台冗余服务器上都运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1311440876'

notice: /Stage[main]/Heartbeat/Package[heartbeat]/ensure: creat
notice: /Stage[main]/Heartbeat/File[/etc/ha.d/authkeys]/ensure:
defined content as '{md5}e908c869aabe519aa69acc9e51da3399'

info: /Stage[main]/Heartbeat/File[/etc/ha.d/authkeys]: Scheduli
refresh of Exec[reload-heartbeat]

notice: /Stage[main]/Heartbeat/File[/etc/ha.d/ha.cf]/ensure:
defined content as '{md5}a8d3fdd62a1172cdff150fc1d86d8a6b'

info: /Stage[main]/Heartbeat/File[/etc/ha.d/ha.cf]: Scheduling
refresh of Exec[reload-heartbeat]

notice: /Stage[main]/Heartbeat/File[/etc/ha.d/haresources]/ensu
defined content as '{md5}0f25aefe7f6c4c8e81b3bb6c86a42d60'

info: /Stage[main]/Heartbeat/File[/etc/ha.d/haresources]:
Scheduling refresh of Exec[reload-heartbeat]

notice: /Stage[main]/Heartbeat/Exec[reload-heartbeat]: Triggere
'refresh' from 3 events

notice: Finished catalog run in 27.01 seconds
```

7. 在主服务器节点上，检查它的资源：

```
# cl_status rscstatus -m
This node is holding all resources.
```

8. 在辅助服务器节点上，你应该可以看到如下信息：

```
# cl_status rscstatus -m
This node is holding none resources.
```

9. 在主节点上禁用 Heartbeat 服务：

```
# service heartbeat stop
```

10. 辅助节点现在应该接管了资源：

```
# cl_status rscstatus -m
This node is holding all resources.
```

工作原理

两台服务器上都运行了心跳守护进程，彼此监听心跳信号。 如果主服务器检测到辅助服务器已宕机，什么也不会发生。 而相反地，如果辅助服务器检测到主服务器已宕机，它就接管 IP 地址。 当主服务器恢复运行后，辅助服务器将再次放弃此地址，重新由主服务器接管 IP 地址 （如果 auto_failback 设置成了 on）。 在某些情况下，例如：如果你在主数据库服务器和从数据库服务器之间共享 IP 地址， 你可能不希望这种行为发生，在这种情况下应该将 auto_failback 设置成了 off。

更多用法

现在你有一个共享的 IP 地址（真是名不副实，因为这个地址不是“共享”的，而是在两个服务器之间切换）， 你可以用这个地址来提供高可用性的服务。例如，如果服务器是被托管的 web 站点， 你应该为 web 站点设置 DNS 记录指向这个共享的 IP 地址。 当主服务器宕机时，辅助服务器将接管 IP 地址并继续响应基于此地址的 HTTP 请求。

> 如果你正在使用 SSL 站点，你需要配置基于共享 IP 地址的 SSL 虚拟主机，否则将不能响应基于这个 IP 的 HTTPs 请求。

另外，如果这个 web 站点使用了 sessions，主服务器上的任何 sessions 在故障转移后将会丢失， 除非 sessions 存储在一个分离的共享数据库中。

共享 IP 地址也是实现双路冗余负载均衡器的一种好方式（参考 使用 HAProxy 为多个 web 服务器实现负载均衡 一节）。你也可以使用这种方式为 Puppetmaster 主机提供冗余服务。在 Puppet Labs 站点给出了一个合适的模式：http://projects.puppetlabs.com/projects/1/wiki/High_Availability_Patterns 。

管理 NFS 服务和文件共享

> There are three kinds of death in this world. There's heart death, there's brain death, and there's being off the network.
>
> — Guy Almes

网络文件系统（**Network File System**，**NFS**）是一种从远程服务器挂装共享目录的方法。 例如：一批 web 服务器可以挂装同一个 NFS 共享为客户供应静态资产文件，比如图片文件和 CSS 文件。 尽管 NFS 是一种比较旧的技术，但它仍被广泛使用着， 所以本处方将向你展示如何创建一个 NFS 服务器以及如何通过它来共享文件。

操作步骤

1. 创建一个 nfs 模块：

```
# mkdir /etc/puppet/modules/nfs
# mkdir /etc/puppet/modules/nfs/manifests
```

2. 使用如下内容创建 /etc/puppet/modules/nfs/manifests/init.pp 文件：

```
class nfs {
    package { "nfs-kernel-server": ensure =&gt; installed }

    service { "nfs-kernel-server":
        ensure     =&gt; running,
        enable     =&gt; true,
        hasrestart =&gt; true,
        require    =&gt; Package["nfs-kernel-server"],
    }

    file { "/etc/exports.d":
        ensure =&gt; directory,
    }

    exec { "update-etc-exports":
        command     =&gt; "/bin/cat /etc/exports.d/* &gt;/etc/e
        notify      =&gt; Service["nfs-kernel-server"],
        refreshonly =&gt; true,
    }

    define share( $path, $allowed, $options = "" ) {
        include nfs
        file { $path:
            ensure =&gt; directory,
        }

        file { "/etc/exports.d/${name}":
            content =&gt; "${path}  ${allowed}(${options})\n",
            notify  =&gt; Exec["update-etc-exports"],
        }
    }
}
```

3. 在你需要输出 NFS 共享的节点上添加如下代码（修改 IP 地址范围以适应你的网络）：

```
nfs::share { "data":
    path    =&gt; "/data",
    allowed =&gt; "10.0.2.0/24",
    options =&gt; "rw,sync,no_root_squash",
}

nfs::share { "data2":
    path    =&gt; "/data2",
    allowed =&gt; "10.0.2.0/24",
    options =&gt; "rw,sync,no_root_squash",
}
```

4. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1311526219'
notice: /Stage[main]/Nfs/Package[nfs-kernel-server]/ensure:

created
notice: /Stage[main]/Nfs/Service[nfs-kernel-server]/ensure: ens
changed 'stopped' to 'running'

notice: /Stage[main]//Node[cookbook]/Nfs::Share[data2]/File[/
data2]/ensure: created

notice: /Stage[main]//Node[cookbook]/Nfs::Share[data2]/File[/et
exports.d/data]/ensure: defined content as '{md5}408f8b40815ff4
ec2f324ca7eafc4'

info: /Stage[main]//Node[cookbook]/Nfs::Share[data]/File[/etc/
exports.d/data]: Scheduling refresh of Exec[update-etc-exports]

notice: /Stage[main]//Node[cookbook]/Nfs::Share[data2]/
File[/etc/exports.d/data2]/ensure: defined content as '{md5}
ec2f324ca7eafc4408f8b40815ff4b6e'

info: /Stage[main]//Node[cookbook]/Nfs::Share[data2]/File[/etc/
exports.d/data2]: Scheduling refresh of Exec[update-etc-exports

notice: /Stage[main]/Nfs/Exec[update-etc-exports]: Triggered
'refresh' from 2 events

info: /Stage[main]/Nfs/Exec[update-etc-exports]: Scheduling
refresh of Service[nfs-kernel-server]

notice: /Stage[main]/Nfs/Service[nfs-kernel-server]/ensure: ens
changed 'stopped' to 'running'

notice: /Stage[main]/Nfs/Service[nfs-kernel-server]: Triggered
'refresh' from 1 events

notice: Finished catalog run in 3.13 seconds
```

5. 通过在另一个服务器上挂装共享测试 NFS 的输出配置：

```
# mkdir /mnt/data
# mount cookbook:/data /mnt/data
# ls /mnt/data
```

工作原理

nfs 类用于安装并启动 nfs-kernel-server 服务，此服务监听网络文件共享的连接。还定义了一个 nfs::share 资源，你可以在配置清单中的任何位置使用它输出一个 NFS 目录：

```
nfs::share { "data":
    path    => "/data",
    allowed => "10.0.2.0/24",
    options => "rw,sync,no_root_squash",
}
```

此资源的名字是任何你想要给出的一个标签：本例中为 data。path 用于指定要共享的目录。 allowed 参数可以使用一个 CIDR 网络地址（正向本例中的 10.0.2.0/24）、 一个 IP 地址、一个主机名或一个用空格间隔的 IP 地址或主机名列表。 NFS 服务器仅允许由指定的主机远程挂装此资源。

options 参数用于指定 NFS 的挂装选项（这些参数将出现在 /etc/exports 文件中，可使用命令 man exports 查看这些选项的精确细节）。

请注意，我们使用了与 rsyncd.conf 例子相同的片断模式。 nfs::share 的任何一个实例都会创建 /etc/exports.d 目录下的一个配置文件片段， 这会触发一个 exec 资源将所有的配置文件片段串连成一个 /etc/exports 配置文件并通知 NFS 服务应用配置的改变。

更多用法

NFS 共享只能应用于应用程序的非关键性数据，因为 NFS 服务器会造成单点故障。 你可以使用集群文件系统解决关键性数据的共享，应该考虑使用 GlusterFS 替代 NFS。

使用 HAProxy 为多个 web 服务器实现负载均衡

> The inside of a computer is as dumb as hell but it goes like mad!
>
> — Richard Feynman

曾几何时，为缓慢的 Web 服务器加速的方式就是增加更多的 CPU 核心。 我记得一个老板买了一台 24核的 Sun 工作站，它是个尺寸接近悍马的怪物， 以至于我们不得不扩修数据中心的大门才能将其放入。

时至今日，扩展 Web 站点规模仍旧是添加 CPU 核心的问题，但与从前不同的是：要么他们自己拿来若干粉色的 PC 机作为服务器使用； 要么从云服务提供商那里租用服务器作为计算资源使用。 为了对所有这些核心进行分组，从而一起服务于单一的 Web 站点， 我们要使用负载均衡器（load balancer）。

曾经，负载均衡器是坐落在机架上的一个耗资八万美元的大盒子。 尽管你现在仍旧可以购买这种负载均衡器，但对于大多数组织来说， 使用 Linux 商品服务器的软件负载均衡解决方案可以大大削减成本。

对大多数管理员来说，**HAProxy** 是软件负载均衡器的一种选择：快速、强大、高度可配置。 在下面的处方中，将向你展示如何创建一个 HAProxy 服务器为两个现有的后端服务器实现对其 web 请求的负载均衡。

操作步骤

1. 创建 loadbalancer 模块：

   ```
   # mkdir /etc/puppet/modules/loadbalancer
   # mkdir /etc/puppet/modules/loadbalancer/manifests
   # mkdir /etc/puppet/modules/loadbalancer/files
   ```

2. 使用如下内容创建 /etc/puppet/modules/loadbalancer/manifests/init.pp 文件：

```
class loadbalancer {
  package { "haproxy": ensure =&gt; installed }

  file { "/etc/default/haproxy":
    source  =&gt; "puppet:///modules/loadbalancer/haproxy.defau
    require =&gt; Package["haproxy"],
  }

  service { "haproxy":
    ensure  =&gt; running,
    enable  =&gt; true,
    require =&gt; Package["haproxy"],
  }

  file { "/etc/haproxy/haproxy.cfg":
    source  =&gt; "puppet:///modules/loadbalancer/haproxy.cfg",
    require =&gt; Package["haproxy"],
    notify  =&gt; Service["haproxy"],
  }
}
```

3. 使用如下内容创建 /etc/puppet/modules/loadbalancer/files/haproxy.defaults 文件：

```
# Don't edit this file - it's managed by Puppet
# Set ENABLED to 1 if you want the init script to start haproxy
ENABLED=1
# Add extra flags here.
#EXTRAOPTS="-de -m 16"
```

4. 使用如下内容创建 /etc/puppet/modules/loadbalancer/files/haproxy.cfg 文件。在 myapp 部分中，用你的后端服务器 IP 地址替换每个 server 行中的 IP 地址， 并用你的后端服务器监听端口替换端口 :8000。

```
global
        daemon
        user haproxy
        group haproxy
        pidfile /var/run/haproxy.pid

defaults
        log         global
        stats       enable
        mode        http
        option      httplog
        option      dontlognull
        option      dontlog-normal
        retries     3
        option      redispatch
        contimeout 4000
        clitimeout 60000
        srvtimeout 30000

listen stats :8080
       mode  http
       stats uri /
       stats auth haproxy:topsecret

listen myapp 0.0.0.0:80
       balance leastconn
       server  myapp1 10.0.2.30:8000    check maxconn 100
       server  myapp2 10.0.2.40:8000    check maxconn 100
```

5. 在你的 HAProxy 节点中包含如下代码：

```
include loadbalancer
```

6. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1311616315'

notice: /Stage[main]/Loadbalancer/Package[haproxy]/ensure: ensu
changed 'purged' to 'present'

--- /etc/haproxy/haproxy.cfg 2009-11-06 17:59:44.000000000

+0000
+++ /tmp/puppet-file20110725-16369-1b85cr8-0 2011-07-25
18:09:03.749146699 +0000
```

```
@@ -1,86 +1,28 @@
-# this config needs haproxy-1.1.28 or haproxy-1.2.1
...

info: /Stage[main]/Loadbalancer/File[/etc/haproxy/haproxy.
cfg]: Filebucketed /etc/haproxy/haproxy.cfg to puppet with sum
c3bfb0c86138552475dea458e8ab36f3

notice: /Stage[main]/Loadbalancer/File[/etc/haproxy/haproxy.cfg
content: content changed '{md5}c3bfb0c86138552475dea458e8ab36f3
to '{md5}fa5fac3cf31f043f0120d0d45cef3f54'

info: /Stage[main]/Loadbalancer/File[/etc/haproxy/haproxy.cfg]:
Scheduling refresh of Service[haproxy]

notice: /Stage[main]/Loadbalancer/Service[haproxy]/ensure: ensu
changed 'stopped' to 'running'

notice: /Stage[main]/Loadbalancer/Service[haproxy]: Triggered
'refresh' from 1 events

--- /etc/default/haproxy 2009-11-06 17:59:21.000000000 +0000

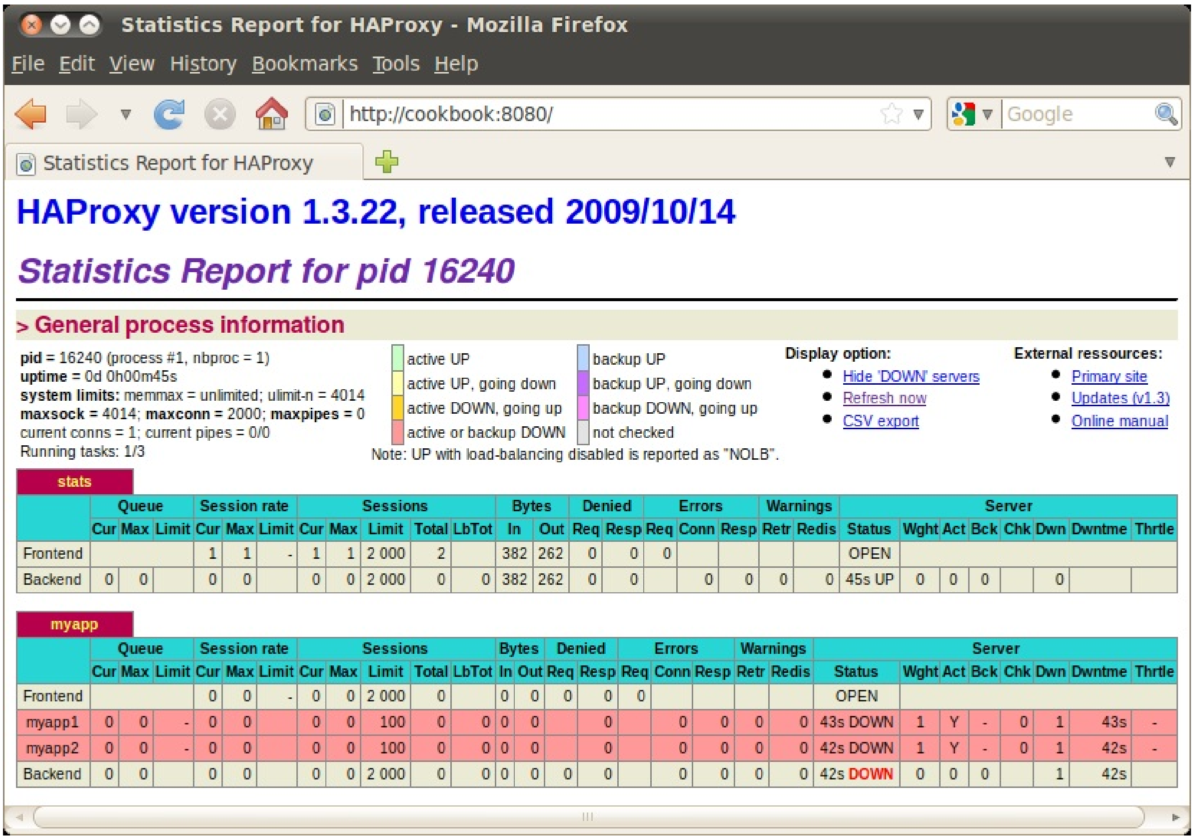
+++ /tmp/puppet-file20110725-16369-1ndfrti-0 2011-07-25
18:09:05.749136866 +0000
@@ -1,4 +1,5 @@

 # Set ENABLED to 1 if you want the init script to start haprox
-ENABLED=0
+ENABLED=1
 # Add extra flags here.
 #EXTRAOPTS="-de -m 16"
+

notice: /Stage[main]/Loadbalancer/File[/etc/default/haproxy]/
content: content changed '{md5}a1f2deb7c7a10e55dc7c971a2288f5d4
to '{md5}2217d74d66bd72630268598b1f11f173'

notice: Finished catalog run in 22.21 seconds
```

7. 在你的浏览器中检查 HAProxy 的 **stats** 界面确保一切工作正常（注意我的 **Backend** 服务器显示的是 **DOWN** ，因为这些虚拟机还没有运行。当我启动它们之后，HAProxy 将会自动检测并重新标记它们的状态）。



工作原理

haproxy 守护进程监听进入的请求并将其分发到一组后端服务器（本例中是 **myapp1** 和 **myapp2**）。如果一个后端服务器已超载，HAProxy 将避免向其发送更多的流量直至它恢复。这有助于防止单台 Web 服务器因为超载（排队等待的不能被处理的请求越来越多）而造成的响应速度大幅放缓。如果一台后端服务器宕机，HAProxy 将不会为其分发任何请求，直至其重新可用。

stats 界面会显示：你的后端服务器如何执行，有多少会话正在处理，HAProxy 是否将后端服务器标记成了 **UP** 或 **DOWN**，等信息。

更多用法

如果你想添加更多的后端服务器以处理不断增长的需求，只需在 haproxy.cfg 中添加更多的 server 行。如果你发现现有的后端服务器响应速度越来越慢，请适当减少每个服务器的 maxconn 值。HAProxy 有大量值得探索的配置参数，参考 HAProxy 的文档站点 http://haproxy.1wt.eu/#docs 。

如果你需要提供 SSL 的能力，可以将 Nginx 放在 HAProxy 的前端进行处理。

尽管 HAProxy 经常用于 Web 服务器，但它可以代理很多服务，不仅仅是 HTTP。它可以处理任何 TCP 流量，所以你可以使用 HAProxy 为 MySQL 服务器、SMTP 服务器、视频服务器以及任何你想要的服务器实现负载均衡。

使用 iptables 管理防火墙

> Programming can be fun, so can cryptography; however they should not be combined.
>
> — Kreitzberg and Shneiderman

C 编程语言被形容为“只写”的语言；它是如此的简洁、高效， 甚至你自己读自己写过的代码都可能很难理解。 同样地，Linux 内核内置的包过滤防火墙的 iptables 的配置也是如此。 一条原始的 iptables 命令规则看上去像这样：

```
iptables -A INPUT -d 10.0.2.15/32 -p tcp -m tcp --dport 80 -j ACCEP
```

除非你会因为掌握了命令行的这些似乎毫无意义的字符串而获得男子气概 （诚然这是 UNIX 系统管理员的职业病）， 否则，能够以更加象征性和可读性的方式来表达防火墙规则是更好的选择。 Puppet 在这方面可以为我们提供帮助，因为我们可以用它对 iptables 的实现细节进行抽象， 并通过参考管理员所控制的服务角色来定义防火墙规则，例如：

```
iptables::role { "web-server": }
```

准备工作

你需要我们在第 5 章 为配置文件添加配置行 一节中创建的 append_if_no_such_line 函数。

操作步骤

1. 创建一个 iptables 模块：

   ```
   # mkdir /etc/puppet/modules/iptables
   # mkdir /etc/puppet/modules/iptables/manifests
   # mkdir /etc/puppet/modules/iptables/files
   ```

2. 使用如下内容创建 /etc/puppet/modules/iptables/manifests/init.pp 文件：

   ```
   class iptables {
       file { [ "/root/iptables",
                "/root/iptables/hosts",
                "/root/iptables/roles" ]:
           ensure =&gt; directory,
       }
   ```

```
    file { "/root/iptables/roles/common":
        source =&gt; "puppet:///modules/iptables/common.role",
        notify =&gt; Exec["run-iptables"],
    }

    file { "/root/iptables/names":
        source =&gt; "puppet:///modules/iptables/names",
        notify =&gt; Exec["run-iptables"],
    }

    file { "/root/iptables/iptables.sh":
        source =&gt; "puppet:///modules/iptables/iptables.sh",
        mode   =&gt; "755",
        notify =&gt; Exec["run-iptables"],
    }

    file { "/root/iptables/hosts/${hostname}":
        content =&gt; "export MAIN_IP=${ipaddress}\n",
        replace =&gt; false,
        require =&gt; File["/root/iptables/hosts"],
        notify  =&gt; Exec["run-iptables"],
    }

    exec { "run-iptables":
        cwd         =&gt; "/root/iptables",
        command     =&gt; "/usr/bin/test -f hosts/${hostname} &
/root/iptables/iptables.sh && /sbin/iptables-save &gt;
/etc/iptables.rules",
        refreshonly =&gt; true,
    }

    append_if_no_such_line { "restore iptables rules":
        file =&gt; "/etc/network/interfaces",
        line =&gt; "pre-up iptables-restore &lt; /etc/iptables.
    }

    define role() {
        include iptables

        file { "/root/iptables/roles/${name}":
            source  =&gt; "puppet:///modules/iptables/${name}.r
            replace =&gt; false,
            require =&gt; File["/root/iptables/roles"],
            notify  =&gt; Exec["run-iptables"],
        }

        append_if_no_such_line { "${name} role":
            file    =&gt; "/root/iptables/hosts/${hostname}",
            line    =&gt; ". `dirname \$0`/roles/${name}",
            require =&gt; File["/root/iptables/hosts/${hostname
            notify =&gt; Exec["run-iptables"],
        }
    }
```

```
}
```

3. 使用如下内容创建 /etc/puppet/modules/iptables/files/iptables.sh 文件：

```
# Server names and ports
. `dirname $0`/names

# Interfaces (override in host-specific file if necessary)
export EXT_INTERFACE=eth0

# Flush and remove all chains
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -F
iptables -X

# Allow all traffic on loopback interface
iptables -I INPUT 1 -i lo -j ACCEPT
iptables -I OUTPUT 1 -o lo -j ACCEPT

# Allow established and related connections
iptables -I INPUT 2 -m state --state ESTABLISHED,RELATED -j ACC
iptables -I OUTPUT 2 -m state --state ESTABLISHED,RELATED -j AC

# Include machine specific settings
HOST_RULES=`dirname $0`/hosts/`hostname -s`
[ -f ${HOST_RULES} ] && . ${HOST_RULES}
[ "${MAIN_IP}" == "" ] && ( echo No MAIN_IP was set, \
 please set the primary IP address in ${HOST_RULES}. ; exit 1 )

# Include common settings
. `dirname $0`/roles/common

# Drop all non-matching packets
iptables -A INPUT -j LOG --log-prefix "INPUT: "
iptables -A INPUT -j DROP
iptables -A OUTPUT -j LOG --log-prefix "OUTPUT: "
iptables -A OUTPUT -j DROP

echo -e "Test remote login and then:\n iptables-save \
 &gt;/etc/iptables.rules"
```

4. 使用如下内容创建 /etc/puppet/modules/iptables/files/names 文件：

```
# Servers
export PUPPETMASTER=10.0.2.15

# Well-known ports
export DNS=53
export FTP=21
export GIT=9418
export HEARTBEAT=694
export IMAPS=993
export IRC=6667
export MONIT=2828
export MYSQL=3306
export MYSQL_MASTER=3307
export NRPE=5666
export NTP=123
export POSTGRES=5432
export PUPPET=8140
export RSYNCD=873
export SMTP=25
export SPHINX=3312
export SSH=22
export STARLING=3307
export SYSLOG=514
export WEB=80
export WEB_SSL=443
export ZABBIX=10051
```

5. 使用如下内容创建 /etc/puppet/modules/iptables/files/common.role 文件：

```
# Common rules for all hosts
iptables -A INPUT -p tcp -m tcp -d ${MAIN_IP} --dport ${SSH} -j

iptables -A INPUT -p ICMP --icmp-type echo-request -j ACCEPT
iptables -A OUTPUT -p ICMP --icmp-type echo-request -j ACCEPT

iptables -A OUTPUT -p tcp --dport ${SSH} -j ACCEPT
iptables -A OUTPUT -p tcp --dport ${SMTP} -j ACCEPT
iptables -A OUTPUT -p udp --dport ${NTP} -j ACCEPT
iptables -A OUTPUT -p tcp --dport ${NTP} -j ACCEPT
iptables -A OUTPUT -p udp --dport ${DNS} -j ACCEPT
iptables -A OUTPUT -p tcp --dport ${WEB} -j ACCEPT
iptables -A OUTPUT -p tcp --dport ${WEB_SSL} -j ACCEPT
iptables -A OUTPUT -p tcp -d ${PUPPETMASTER} --dport ${PUPPET}
iptables -A OUTPUT -p tcp --dport ${MYSQL} -j ACCEPT

# Drop some commonly probed ports
iptables -A INPUT -p tcp --dport 23 -j DROP # telnet
iptables -A INPUT -p tcp --dport 135 -j DROP # epmap
iptables -A INPUT -p tcp --dport 139 -j DROP # netbios
iptables -A INPUT -p tcp --dport 445 -j DROP # Microsoft DS
iptables -A INPUT -p udp --dport 1433 -j DROP # SQL server
iptables -A INPUT -p tcp --dport 1433 -j DROP # SQL server
iptables -A INPUT -p udp --dport 1434 -j DROP # SQL server
iptables -A INPUT -p tcp --dport 1434 -j DROP # SQL server
iptables -A INPUT -p tcp --dport 2967 -j DROP # SSC-agent
```

6. 使用如下内容创建 /etc/puppet/modules/iptables/files/web-server.role 文件：

```
# Access to web
iptables -A INPUT -p tcp -d ${MAIN_IP} --dport ${WEB} -j ACCEPT

# Send mail from web applications
iptables -A OUTPUT -p tcp --dport ${SMTP} -j ACCEPT
```

7. 使用如下内容创建 /etc/puppet/modules/iptables/files/puppet-server.role 文件：

```
# Access to puppet
iptables -A INPUT -p tcp -d ${MAIN_IP} --dport ${PUPPET} -j ACC
```

8. 在你的 Puppetmaster 节点上包含如下内容：

```
iptables::role { "web-server": }
iptables::role { "puppet-server": }
```

9. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1311682880'

notice: /Stage[main]/Iptables/File[/root/iptables]/ensure: crea
notice: /Stage[main]/Iptables/File[/root/iptables/names]/ensure
defined content as '{md5}9bb004a7d2c6d70616b149d044c22669'

info: /Stage[main]/Iptables/File[/root/iptables/names]: Schedul
refresh of Exec[run-iptables]

notice: /Stage[main]/Iptables/File[/root/iptables/hosts]/ensure
created

notice: /Stage[main]/Iptables/File[/root/iptables/hosts/cookboo
ensure: defined content as '{md5}d00bc730514bbb74cdef3dad70058a

info: /Stage[main]/Iptables/File[/root/iptables/hosts/cookbook]
Scheduling refresh of Exec[run-iptables]

notice: /Stage[main]//Node[cookbook]/Iptables::Role[web-server]
Append_if_no_such_line[web-server role]/Exec[/bin/echo '. `dirn
$0`/roles/web-server' &gt;&gt; '/root/iptables/hosts/cookbook']
executed successfully

info: /Stage[main]//Node[cookbook]/Iptables::Role[web-server]/
Append_if_no_such_line[web-server role]/Exec[/bin/echo '. `dirn
$0`/roles/web-server' &gt;&gt; '/root/iptables/hosts/cookbook']
Scheduling refresh of Exec[run-iptables]

notice: /Stage[main]//Node[cookbook]/Iptables::Role[puppetserve
Append_if_no_such_line[puppet-server role]/Exec[/bin/echo
'. `dirname $0`/roles/puppet-server' &gt;&gt; '/root/iptables/h
cookbook']/returns: executed successfully

info: /Stage[main]//Node[cookbook]/Iptables::Role[puppet-server
Append_if_no_such_line[puppet-server role]/Exec[/bin/echo '.
`dirname $0`/roles/puppet-server' &gt;&gt; '/root/iptables/host
cookbook']: Scheduling refresh of Exec[run-iptables]

notice: /Stage[main]/Iptables/File[/root/iptables/roles]/ensure
created

notice: /Stage[main]//Node[cookbook]/Iptables::Role[puppetserve
File[/root/iptables/roles/puppet-server]/ensure: defined
```

```
content as '{md5}c30a13f7792525c181e14e78c9a510cd'

info: /Stage[main]//Node[cookbook]/Iptables::Role[puppet-server
File[/root/iptables/roles/puppet-server]: Scheduling refresh of
Exec[run-iptables]

notice: /Stage[main]//Node[cookbook]/Iptables::Role[web-server]
File[/root/iptables/roles/web-server]/ensure: defined content a
'{md5}11e5747cb2737903ffc34133f5fe2452'

info: /Stage[main]//Node[cookbook]/Iptables::Role[web-server]/
File[/root/iptables/roles/web-server]: Scheduling refresh of
Exec[run-iptables]

notice: /Stage[main]/Iptables/File[/root/iptables/roles/common]
ensure: defined content as '{md5}116f57d4e31f3e0b351da6679dca15

info: /Stage[main]/Iptables/File[/root/iptables/roles/common]:
Scheduling refresh of Exec[run-iptables]

notice: /Stage[main]/Iptables/File[/root/iptables/iptables.sh]/
ensure: defined content as '{md5}340ff9fb5945e9fc7dd78b21f45dd8

info: /Stage[main]/Iptables/File[/root/iptables/iptables.sh]:
Scheduling refresh of Exec[run-iptables]

notice: /Stage[main]/Iptables/Exec[run-iptables]: Triggered
'refresh' from 8 events

notice: /Stage[main]/Iptables/Append_if_no_such_line[restore
iptables rules]/Exec[/bin/echo 'pre-up iptables-restore &lt; /e
iptables.rules' &gt;&gt; '/etc/network/interfaces']/returns: ex
successfully

notice: Finished catalog run in 4.86 seconds
```

10. 检查要求的规则是否已被安装：

```
# iptables -nL
Chain INPUT (policy ACCEPT)
target prot opt source      destination
ACCEPT all  --  0.0.0.0/0  0.0.0.0/0
ACCEPT all  --  0.0.0.0/0  0.0.0.0/0    state RELATED,ESTABLISH
ACCEPT tcp  --  0.0.0.0/0  10.0.2.15    tcp dpt:80
ACCEPT tcp  --  0.0.0.0/0  10.0.2.15    tcp dpt:8140
ACCEPT tcp  --  0.0.0.0/0  10.0.2.15    tcp dpt:22
ACCEPT icmp --  0.0.0.0/0  0.0.0.0/0    icmp type 8
DROP   tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:23
DROP   tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:135
DROP   tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:139
DROP   tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:445
DROP   udp  --  0.0.0.0/0  0.0.0.0/0    udp dpt:1433
DROP   tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:1433
DROP   udp  --  0.0.0.0/0  0.0.0.0/0    udp dpt:1434
DROP   tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:1434
DROP   tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:2967
LOG    all  --  0.0.0.0/0  0.0.0.0/0    LOG
 flags 0 level 4 prefix `INPUT: '
DROP   all  --  0.0.0.0/0  0.0.0.0/0

Chain FORWARD (policy ACCEPT)
target prot opt source destination
Chain OUTPUT (policy ACCEPT)
target prot opt source      destination
ACCEPT all  --  0.0.0.0/0  0.0.0.0/0
ACCEPT all  --  0.0.0.0/0  0.0.0.0/0    state RELATED,ESTABLISH
ACCEPT tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:25
ACCEPT icmp --  0.0.0.0/0  0.0.0.0/0    icmp type 8
ACCEPT tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:22
ACCEPT tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:25
ACCEPT udp  --  0.0.0.0/0  0.0.0.0/0    udp dpt:123
ACCEPT tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:123
ACCEPT udp  --  0.0.0.0/0  0.0.0.0/0    udp dpt:53
ACCEPT tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:80
ACCEPT tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:443
ACCEPT tcp  --  0.0.0.0/0  10.0.2.15    tcp dpt:8140
ACCEPT tcp  --  0.0.0.0/0  0.0.0.0/0    tcp dpt:3306
LOG    all  --  0.0.0.0/0  0.0.0.0/0    LOG
flags 0 level 4 prefix `OUTPUT: '
DROP   all  --  0.0.0.0/0  0.0.0.0/0
```

工作原理

为了创建一套合适的防火墙规则，我们需要知道节点的主 IP 地址以及其运行了哪些服务。 我们还需要添加一些所有的机器都要设置的共同规则（例如，允许 SSH），并运行一系列的 iptables 命令以激活我们已经生成的规则。 之后我们还要

保存这些规则，以便使这些规则可以在开机时恢复执行。 下面介绍所有的这一切是如何完成的。

首先，我们创建一个 names 文件为常用的端口定义 shell 变量。 这意味着，当我们定义防火墙规则时，可以引用变量，例如对于 MySQL 服务可以使用变量 ${MYSQL} 取代数值端口号 3306。

common.role 文件包含了一些对所有机器都有用的规则。编辑这个文件以适应你自己的所有机器 （例如，你可能仅允许从指定的 IP 范围访问 SSH）。

web-server.role 和 puppet-server.role 文件包含了两个特定角色的规则。 你可以添加更多的文件用于定义你的网络中所需的众多角色：例如，数据库服务器、应用服务器、 DNS 服务器，等等。文件中所有的规则都具有如下的格式：

```
iptables -A INPUT -p tcp -d ${MAIN_IP} --dport ${WEB} -j ACCEPT
```

通常，你只需要修改 ${WEB} 这部分：将其替换为另一个端口名（定义在 names 文件中） 的变量引用（例如 ${POSTGRES}）。如果你需要更多的端口名，在 names 文件中添加相应的定义。

iptables.sh 脚本读取其他的所有文件并执行规则要求的 iptables 命令。 每当相关文件有任何改变，Puppet 就执行这个脚本，因此要想刷新防火墙， 你需要做的工作就是改变相关的配置并运行 Puppet。

Puppet 还会将当前的规则集保存在 /etc/iptables.rules 文件中。 为了让机器重启后加载规则集文件，Puppet 在 /etc/network/interfaces 文件中添加了如下一行：

```
pre-up iptables-restore < /etc/iptables.rules
```

所有这一切意味着，在相关的模块中（例如 apache 模块）， 你只要简单地包含如下一行即可创建相应的防火墙：

```
iptables::role { "web-server": }
```

一旦防火墙被激活，任何不符合规则的数据包将被阻止并记录到 /var/log/messages 日志文件。 检查这个文件，以帮助解决防火墙的任何问题。

更多用法

如果在你的规则中引用了某些特定的机器（例如，你的监控服务器）， 可以在 names 文件中添加如下的定义：

```
MONITOR=10.0.2.15
```

然后在适当的位置（例如 common.role 文件中），你可以允许来自这台机器的访问， 例如，允许来自监控服务器对指定主机 NRPE 端口的访问规则如下：

```
iptables -A INPUT -p tcp -m tcp -d ${MAIN_IP} -s ${MONITOR} --dport
```

你也可以用这种方法指定数据库服务器，以及在 .role 文件中需要引用特定的 IP 地址、 网络地址和地址范围等情况。

像这样动态生成防火墙规则集对于云基础设施是非常有用的， 云中的服务器列表会因为节点的创建和销毁而不断地变化。 对于需要触发防火墙重建的任何资源，你只要在此资源中添加如下代码即可：

```
notify => Exec["run-iptables"],
```

你可能有一个由版本控制系统维护的或通过 **cloud API** （例如，Rackspace 或 Amazon EC2） 自动更新的“主服务器列表（master server list）”。 可以在 Puppet 中将这个列表定义成一个 file 资源，通过在此资源中使用 notify 参数即可触发防火墙的重建，所以每次当你检入（check in）主服务器列表的变化， 每台机器上运行的 Puppet 将相应地更新其防火墙。

当然，这种高度的自动化意味着你需要对你检入的内容格外小心， 因为任何错误都可能会导致整个基础设施离线。

测试变更的一种好方法是对用于测试的 Puppet 配置清单使用一个单独的 Git 分支， 在分支中仅将变更应用到一到两台服务器。一旦你验证了变更的正确性， 就可以将其合并到主分支并回滚到主分支。

管理 Amazon 的 EC2 实例

> The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry.
>
> — Henry Petroski

如果你觉得你的电脑近年来变得比较慢，这可能是正确的。 对于大多数应用程序而言，你已不必将所有的计算能力（computing power） 全部挤进你办公桌下的一个米色盒子里。 为了解决这个问题，计算能力已经成为一种可以在网上购买的商品。

亚马逊不只是卖书了：他们也卖首饰、摩托车、叶鼓风机，以及对我们当前目的有用的计算能力。 你可以用信用卡签约使用 **Amazon Web Services**，并根据你的需要创建众多的服务器实例， 每个服务器基于小时计费。如果你只是想试水，可以运行一个免费最长一年的微型实例。 如果你正在寻找将你的一部分基础设施迁移到公共云的方案，这是一个不错的实验方式。

本处方将向你展示一个使用 Puppet 创建并自动供应一个 **EC2** 实例的简单方法。尽管还可以使用更强大的方式实现，包括使用 MCollective，但作为教学目的， 我们将只做最低限度的必要的工作：让一个实例运行起来并应用 Puppet 配置清单。一旦你获得了基本思路，你就可以以此作为基础加入你自己的改良和改进。

准备工作

你需要一个 **Amazon Web Services**（**AWS**）账号。如果还没有，请到 http://aws-portal.amazon.com/gp/aws/developer/subscription/index.html?productCode=AmazonEC2 注册。

你需要 AWS 访问的 key ID，这个秘密访问 Key 对应于你的 AWS 账号。你可以在这个页面找到它： http://aws-portal.amazon.com/gp/aws/developer/account/index.html?action=access-key 。

你还需要访问 EC2 实例的 SSH 密钥对。为此，你需要如下步骤：

1. 在 https://console.aws.amazon.com/ec2/home 登录 AWS 管理控制台。
2. 选择 **Amazon EC2** 标签，在导航部分的标题 **Network & Security** 上单击 **Key Pairs**。
3. 单击 **Create key pair**，当提示出现后下载 keypair 文件。将此文件保存到一个安全的地方， 并使用如下命令为它设置模式为 0600 的权限：

   ```
   # chmod 600 bitfield.pem
   ```

操作步骤

1. 创建 fog 模块：

```
# mkdir /etc/puppet/modules/fog
# mkdir /etc/puppet/modules/fog/manifests
# mkdir /etc/puppet/modules/fog/files
```

2. 使用如下内容创建 /etc/puppet/modules/fog/manifests/init.pp 文件：

```
class fog {
    package { "fog":
        ensure   =&gt; installed,
        provider =&gt; gem,
    }

    file { "/usr/local/etc/fog_credentials":
        source =&gt; "puppet:///modules/fog/fog_credentials",
    }

    file { "/usr/local/bin/boot-ec2":
        source =&gt; "puppet:///modules/fog/boot-ec2.rb",
        mode   =&gt; "755",
    }

    file { "/usr/local/bin/bootstrap-ec2":
        source =&gt; "puppet:///modules/fog/bootstrap-ec2.sh",
        mode   =&gt; "755",
    }
}
```

3. 使用如下内容创建 /etc/puppet/modules/fog/files/boot-ec2.rb 文件（修改 :private_key_path 参数指向你自己的 AWS 私钥文件）：

```
#!/usr/bin/ruby
require 'rubygems'
require 'fog'

HOSTNAME = 'devbox'
@server = ''
Fog.credentials_path = '/usr/local/etc/fog_credentials'

def command( cmdline )
    puts "Running command: #{cmdline}"
    res = @server.ssh( "sudo #{cmdline}" )[0]
    puts res.stdout
    puts res.stderr
end

def create()
    puts "Bootstrapping instance..."
    connection = Fog::Compute.new( { :provider =&gt; 'AWS' } )
    @server = connection.servers.bootstrap( :key_name =&gt;
                                            'bitfield',
                                            :private_key_path =
                                            '~/bitfield.pem',
                                            :username =&gt; 'ub
    @server.wait_for { ready? }
    @server.reload
    puts "Instance name: #{@server.dns_name}"
    puts "Setting hostname..."
    @server.ssh( "sudo hostname #{HOSTNAME}" )
end

def copy_bootstrap_files()
    puts "Copying bootstrap files..."
    @server.scp( "puppet.tar.gz", "/tmp" )
    @server.scp( "/usr/local/bin/bootstrap-ec2", "/tmp" )
end

def bootstrap()
    puts "Bootstrapping..."
    command( "sudo sh /tmp/bootstrap-ec2" )
end

create()
copy_bootstrap_files()
bootstrap()
```

4. 使用如下内容创建 /etc/puppet/modules/fog/files/bootstrap-ec2.sh 文件：

```
#!/bin/bash
apt-get update
apt-get -y install puppet
apt-get -y install git-core
cd /root
tar xzf /tmp/puppet.tar.gz
puppet --modulepath=/root/puppet/modules \
  /root/puppet/manifests/site.pp
```

5. 使用如下内容创建 /etc/puppet/modules/fog/files/fog_credentials 文件（使用你自己的 AWS 凭证替换相应的值）：

```
:default:
  :aws_access_key_id: AKIAI5RGMC3QRPO3AJWR
  :aws_secret_access_key: iygf2+7SfKV/OlEyrh+otazeVin9G3XXrvJYK
```

6. 添加如下的节点声明，此节点将会应用 EC2 实例：

```
node devbox {
    file { "/etc/motd":
        content =&gt; "Puppet power!\n",
    }
}
```

7. 添加如下代码到一个节点：

```
include fog
```

8. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1313160844'

notice: /Stage[main]/Fog/Package[fog]/ensure: ensure changed
'purged' to 'present'

notice: /Stage[main]/Fog/File[/usr/local/bin/bootstrap-ec2]/
ensure: defined content as '{md5}5bc2ffb3b5aa94b33b17d419625ecb

notice: /Stage[main]/Fog/File[/usr/local/bin/boot-ec2]/ensure:
defined content as '{md5}dadc835c6e52c89cb928d60db7677713'

notice: /Stage[main]/Fog/File[/usr/local/etc/fog_credentials]/
ensure: defined content as '{md5}3b140aedac170bbfcc2837077e03bb

notice: Finished catalog run in 1.67 seconds
```

9. 在你的工作目录中，为分发 EC2 实例创建一个 Puppet 的 tar 包。最简单的方法是对你现有的 Puppet 仓库（对于 git 的裸仓库需要先检出）执行 tar 命令：

```
# cd /etc
# tar czf /tmp/puppet.tar.gz --exclude .git puppet
# cd -
# mv /tmp/puppet.tar.gz .
```

10. 运行 boot-ec2 脚本：

```
# boot-ec2
Bootstrapping instance...
Instance name: ec2-107-20-59-174.compute-1.amazonaws.com
Setting hostname...
Copying bootstrap files...
Bootstrapping...
Running command: sudo sh /tmp/bootstrap.sh
sudo: unable to resolve host devbox
sudo: unable to resolve host devbox
...
notice: //Node[devbox]/File[/etc/motd]/content: defined content
'unknown checksum'
```

11. 登录到该实例检查你的配置清单已被正确应用：

```
# ssh -i bitfield.pem ubuntu@ec2-107-20-59-174.compute-1.amazor
Puppet power!
ubuntu@devbox:~$
```

12. 你已经得到了一个由 Puppet 管理控制的云服务器！ 如果想要创建十个实例，运行此脚本十次即可。 不要忘记使用完毕之后，要在 AWS 管理控制台关闭这些实例（按使用时间付费的哦！）。

工作原理

Fog 是一个用于管理云资源的 Ruby 库，包括 EC2 和其他一些供应商（如 Rackspace）。 尽管你可以使用 Amazon 自己提供的 ec2-tools 脚本启动和管理 EC2 实例， 但使用 Fog 可以轻而易举地将你的实例迁移到另一个供应商， 而且还不需要你为运行 ec2-tools 而安装 Java 以及其他依赖的软件。 对于创建 EC2 基础设施的这两种方法来说，我可以自信的说我更喜欢用 Fog， 尽管事实上它几乎没有文档（而 Amazon 却有很多）。

在 boot-ec2 脚本中，我们用自己的凭证使用 Fog 创建了一个新的 EC2 实例， 并为其传输了一份 Puppet 配置清单的拷贝。 然后我们复制 bootstrap-ec2 脚本，它用于安装 Puppet 以及应用程序的配置清单。

对于本例而言，配置清单相当简单：

```
file { "/etc/motd":
    content => "Puppet power!\n",
}
```

你可以轻松地修改它，比如，与你的生产应用服务器同名。 这对于快速部署位于物理负载平衡器之后的大量应用程序服务器来说是一个好方法， 例如，处理突然飙升的需求。另外，你也可以使用 EC2 实例作为测试服务器或临时服务器 （这完全取决于你自己的需求）。

更多用法

除了你的信用卡强加给你的限制之外，此处的脚本没有对你能部署的 EC2 实例数量做任何限制。 所以你可以尝试修改此处所示的脚本，启用一个由命令行参数设置的实例数量。

你或许想要创建不同类型的实例，例如：Web 服务器、人工队列服务器（queue worker servers）等。 你可以修改启动脚本，携带一个参数指定要启动的实例类型。

此处显示的脚本有一个重要的限制，就是它以 tar 包形式提供了一个包含你的 Puppet 配置清单快照的实例。 显然，当你在 Puppetmaster 上修改你的 Puppet 配置清单后，EC2 实例不受影响，即更改不会被应用到 EC2 实例。 基于简单的目

的，本处方中的例子仅仅使用 Puppet 构建了初始的服务器，它没有运行 Puppet 守护进程也没与 Puppetmaster 服务器联系。

这对于短生存期或仅为指定目的而运行的 EC2 实例往往还是不错的。 如果你需要运行长生存期的服务器，或者需要通过 Puppet 更新 EC2 实例服务器， 就应该修改脚本使实例能与你的 Puppetmaster 服务器取得联系。 要解决的是证书签名问题，例如你可以对证书进行预签名并伴随 bootstrap 脚本一同部署到 EC2 实例。 另外一种方法是，在脚本中通过 SSH 或 MCollective 登录 Puppetmaster 服务器并对实例的证书请求进行签名。 这两种证书签名的机制或简单或复杂，随你选择。

你可能还想使用其他的云服务提供商，例如 Rackspace 或 Linode。 为此，你需要对脚本做轻微地修改。 请参考 Fog 文档获取相关的详细信息，网址为 http://fog.io 。

> 你也可以使用 Puppet 新的云供应商扩展（Cloud Provisioner extension）来管理 EC2 实例； 要获取相关的详细信息请参考 Puppet Labs 的 http://docs.puppetlabs.com/guides/cloud_pack_getting_started.html 页面。

参见本书

- 本章的 使用 Vagrant 管理虚拟机 一节

使用 **Vagrant** 管理虚拟机

> In 1974 computers were oppressive devices in far-off air-conditioned places. Now you can be oppressed in your own living room.
>
> — Ted Nelson

虽然能够在云中部署虚拟机是一个创举，但若能将虚拟机运行在你自己的桌面系统中有时甚至是更方便的， 尤其对于测试来说更是如此。如果每个开发者都有一个克隆自生产系统的运行在自己机器上的虚拟机， 那么实际部署时就不太可能遇到问题。 同样地，每个系统管理员也可以在私人的虚拟机上测试配置管理的变化， 这是一种使错误配置实际影响客户之前捕捉错误的良好方式。

几年前出现的工具（如 VirtualBox 或 VMware）就已经能在桌面系统上创建虚拟机。 然而，**Vagrant** 的到来使桌面云技术真正实现了起飞，它是一个管理和供应虚拟化环境的自动化工具。 Vagrant 是 VirtualBox 的前端工具，它驱动 VirtualBox 实现自动创建虚拟机的过程， 并使用自动化管理配置工具（Chef 或 Puppet）为虚拟机调配所需的资源， 设置网络，端口转发，以及对运行着的虚拟机打包生成映像文件以便其他人使用。

你可以使用 Vagrant 管理你用于开发目的虚拟机，这些虚拟机既可以运行在你自己的桌面上， 也可以运行在一台共享的机器上，比如一台持续集成服务器（Continuous Integration Server）。 例如，你可以使用像 **Jenkins** 那样的 CI 工具启动一个由 Vagrant 管理的虚拟机， 部署你的应用程序，然后在虚拟机里运行测试实验，就好像是在生产环境中一样。

操作步骤

1. 创建一个 vagrant 模块：

   ```
   # mkdir /etc/puppet/modules/vagrant
   # mkdir /etc/puppet/modules/vagrant/manifests
   # mkdir /etc/puppet/modules/vagrant/files
   ```

2. 使用如下内容创建 /etc/puppet/modules/vagrant/manifests/init.pp 文件：

   ```
   class vagrant {
       $virtualbox_deps = [ "libgl1-mesa-glx",
                            "libqt4-network",
                            "libqt4-opengl",
                            "libqtcore4",
                            "libqtgui4",
                            "libsdl1.2debian",
                            "libxmu6",
                            "libxt6",
                            "gawk",
                            "linux-headers-${kernelrelease}" ]
   ```

```
        package { $virtualbox_deps: ensure =&gt; installed }

        exec { "download-virtualbox":
            cwd     =&gt; "/root",
            command =&gt; "/usr/bin/wget http://download.virtua
virtualbox/4.1.0/virtualbox-4.1_4.1.0-73009~Ubuntu~lucid_
i386.deb",
            creates =&gt; "/root/virtualbox-4.1_4.1.0-73009~Ubu
i386.deb",
            timeout =&gt; "-1",
        }

        exec { "install-virtualbox":
            command =&gt; "/usr/bin/dpkg -i /root/virtualbox-4.
73009~Ubuntu~lucid_i386.deb",
            unless  =&gt; "/usr/bin/dpkg -l |/bin/grep virtualb
            require =&gt; [ Exec["download-virtualbox"],
            Package[$virtualbox_deps] ],
        }

        $vagrant_deps = [ "build-essential",
                          "rubygems" ]

        package { $vagrant_deps: ensure =&gt; installed }

        exec { "install-rubygems-update":
            command =&gt; "/usr/bin/gem install -v 1.8.6 rubyge
            unless  =&gt; "/usr/bin/gem -v |/bin/grep 1.8.6",
            require =&gt; Package["rubygems"],
        }

        exec { "run-rubygems-update":
            command =&gt; "/var/lib/gems/1.8/bin/update_rubygem
            unless  =&gt; "/usr/bin/gem -v |/bin/grep 1.8.6",
            require =&gt; Exec["install-rubygems-update"],
        }

        package { "vagrant":
            provider =&gt; gem,
            ensure   =&gt; installed,
            require  =&gt; [ Package["build-essential"],
                          Exec["runrubygems-update"] ],
        }

        define devbox( $vm_user ) {
            include vagrant
            $vm_dir = "/home/${vm_user}/${name}"
            file { [ $vm_dir,
                     "${vm_dir}/data" ]:
                ensure =&gt; directory,
                owner  =&gt; $vm_user,
            }
```

```
            file { "${vm_dir}/Vagrantfile":
                source =&gt; "puppet:///modules/vagrant/devbox.
 Vagrantfile",
                require =&gt; File[$vm_dir],
            }
     }
}
```

3. 使用如下内容创建 /etc/puppet/modules/vagrant/files/devbox.Vagrantfile 文件：

```
Vagrant::Config.run do |config|
  config.vm.box = "lucid32"
  config.vm.box_url = "http://files.vagrantup.com/lucid32.box"
  config.vm.forward_port "http", 80, 8080
  config.vm.share_folder "v-data", "/vagrant_data", "./data"

  config.vm.customize do |vm|
    vm.name = "devbox"
  end

  config.vm.provision :puppet,:module_path =&gt; "puppet/module
  do |puppet|
    puppet.manifests_path = "puppet/manifests"
    puppet.manifest_file = "site.pp"
  end
end
```

4. 在一个你想要运行虚拟机的节点上包含如下代码（将 john 替换为你自己的用户名）：

```
vagrant::devbox { "devbox":
    vm_user =&gt; "john",
}
```

5. 添加一个名为 devbox 的节点：

```
node devbox {
    group { "puppet": ensure =&gt; present }
    file { "/etc/motd":
        content =&gt; "Puppet power!\n",
    }
}
```

6. 运行 Puppet：

```
# puppet agent --test
```

7. 你应该在要运行虚拟机的宿主机上找到用户 john 自家目录下已创建的 devbox 目录。 在此目录中需要拥有一套 Puppet 配置清单的子目录（名为 puppet），既可以从你的 Puppet 仓库检出到名为 puppet 的目录，也可以创建一个名为 puppet 的 符号链接（**symlink**）指向宿主机上已存在的 Puppet 配置清单目录：

```
# cd ~/devbox
# git clone git@github.com:Example/Puppet.git puppet
```

或者

```
# ln -s /etc/puppet ~/devbox/puppet
```

8. 在 devbox 目录中，运行如下命令行：

```
# vagrant up
[default] Box lucid32 was not found. Fetching box from specifie
URL...
[default] Downloading with Vagrant::Downloaders::HTTP...
[default] Downloading box: http://files.vagrantup.com/lucid32.b
[default] Extracting box...
[default] Verifying box...
[default] Cleaning up downloaded box...
[default] Importing base box 'lucid32'...
[default] Matching MAC address for NAT networking...
[default] Clearing any previously set forwarded ports...
[default] Forwarding ports...
[default] -- http: 80 =&gt; 8080 (adapter 1)
[default] -- ssh: 22 =&gt; 2222 (adapter 1)
[default] Creating shared folders metadata...
[default] Running any VM customizations...
[default] Booting VM...
[default] Waiting for VM to boot. This can take a few minutes.
[default] VM booted and ready for use!
[default] Mounting shared folders...
[default] -- v-root: /vagrant
[default] -- v-data: /vagrant_data
[default] -- manifests: /tmp/vagrant-puppet/manifests
[default] Running provisioner: Vagrant::Provisioners::Puppet...
[default] Running Puppet with site.pp...
[default] stdin: is not a tty
[default] notice: /Stage[main]//Node[devbox]/File[/etc/motd]/
ensure: defined content as '{md5}0bdeca690dbb409d48391f3772d389
[default]
[default] notice: /Group[puppet]/ensure: created
[default]
[default] notice: Finished catalog run in 0.36 seconds
[default]
```

登录到 devbox 虚拟主机进行测试：

```
# vagrant ssh
Puppet power!
Last login: Thu Jul 21 13:07:53 2011 from 10.0.2.2
vagrant@devbox:~$ logout
Connection to 127.0.0.1 closed.
```

工作原理

vagrant 类安装 Vagrant 和 VirtualBox 以及所有的依赖。它同时还定义了名为 devbox 的 define，你可以使用它为一台宿主机的多个用户创建 devbox 的多个实例。 devbox 的一个实例如下：

```
vagrant::devbox { "app-foo-devbox":
    vm_user => "john",
}
```

此实例在用户（本例为 john）的家目录下创建一个名为 app-foo-devbox 的 Vagrant 项目目录（此目录包含一个配置文件 Vagrantfile，它指定了一个虚拟机的配置定义）。

当 Vagrant 首次启动虚拟机，它会在项目目录的名为 puppet 的子目录中查找提供给本虚拟机的 Puppet 配置清单。 这可以是你当前 Puppet 工作副本的一个符号链接， 也可以是仅为 devbox 编制的独立的 Puppet 配置清单 （无论你使用哪一种方式，只要 Vagrant 能找到即可）。

一旦虚拟主机已经配置好，它就可以投入使用了。运行 vagrant up 命令即可启动虚拟机； vagrant ssh 命令用于登录虚拟机；vagrant halt 命令用于停止虚拟机的运行。

顺便指出，节点定义中名为 puppet 的 group 资源在 Vagrant 的 Puppet 供应时会引发一个错误， 当你看到本书时可能已经被修复。Vagrant 正处于开发活跃期，所以可能会有一两处无法正常工作： 如有疑问，请查看本节最后的文档链接。

你可能会发现有时虚拟机无法完全启动，Vagrant 只是处于超时等待状态。 这似乎也是由于 Vagrant 的一个错误引起的，当你看到本书时可能已经被修复。如果还没有修复， 你可以在 Vagrantfile 中通过添加如下的代码片段来解决这个问题：

```
config.vm.boot_mode = :gui
```

修改之后重新启动虚拟机。现在虚拟机在 GUI 模式下启动，同时运行了一个控制台窗口。 在此窗口中，以用户名 vagrant（口令为 vagrant）登录，然后运行如下命令：

```
# sudo /etc/init.d/networking restart
```

现在你发现 Vagrant 会完成配置阶段并且 vagrant ssh 命令也会工作正常。

更多用法

在本例中，我们仅对 devbox 配置了一个极其简单的配置清单，它在 /etc/motd 文件中添加了消息。 为了使其更实用，可以让 devbox 提取与你要部署的实际服务器相同的配置清单。例如：

```
node production, devbox {
    include myapp::production
}
```

因此，应用到生产服务器配置的任何改变将同时反映在你用于测试的机器上， 这样就可以在实际部署之前先解决出现的问题，如果你需要进行配置的变化以支持新的功能， 可以首先在虚拟机上测试它，看看是否有什么不正常。

如果你不再使用虚拟机，想要挂起或关闭它，只要运行：

```
# vagrant suspend
```

或

```
# vagrant halt
```

想要完全删除虚拟机，例如你要重新测试供应，运行：

```
# vagrant destroy
```

Vagrant 的维护者为了使其使用简单做了相当多的工作，如果你需要阅读更多关于 Vagrant 的内容请访问其文档站点： http://vagrantup.com/docs/index.html 。

外部工具和 Puppet 生态环境

> Unix is the answer, but only if you phrase the question very carefully.
>
> — Belinda Asbell

在本章中，我们将学习如下内容：

- 创建 Facter 的自定义 fact
- 在运行 Puppet 之前和之后执行命令
- 从 Shell 会话生成 Puppet 配置清单
- 从运行的系统上生成 Puppet 配置清单
- 使用 Puppet Dashboard
- 使用 Foreman
- 使用 MCollective
- 使用公共模块
- 使用外部节点分类器
- 创建自定义的资源类型
- 创建自定义的提供者

Puppet 本身就是一个有用的工具，但组合使用 Puppet 与其他工具和框架可以得到更大的利益。 在本章我们会看到：使用工具（Facter、cft 和 puppet resource）获得 Puppet 所需的数据； 使用工具（Foreman 和 Puppet Dashboard）管理和报告来自 Puppet 的数据。

你还会学到：如何通过创建你自定义的资源类型扩展 Puppet，并在不同的平台上实现自定义类型； 如何使用外部节点分类器脚本整合 Puppet 和数据库（例如：LDAP）； 如何使用来自 Puppet Forge 的公共模块；以及 Puppet 如何与系统管理框架 MCollective 协同工作。

创建 Facter 的自定义 fact

虽然 Facter 内置的 facts 很有用，但实际上添加你自己的 facts 也是很容易的。 例如，如果你的机器位于不同的数据中心或托管提供商，你可以为此目的添加一个 自定义 **fact** 以便让 Puppet 决定是否需要应用一些本地设置（例如，本地 DNS 服务器）。

准备工作

1. 在配置文件 puppet.conf 中开启 pluginsync 选项：

   ```
   [main]
       pluginsync = true
   ```

2. 为 fact 创建一个目录。此目录位于相应的模块目录中，目录名为 lib/facter。例如，你可以使用目录 modules/admin/lib/facter。你创建的任何自定义 facts 都位于此目录下并且 Puppet 会将其同步到客户端。

操作步骤

1. 创建一个名为 hello.rb 的包含如下内容的脚本文件：

   ```
   Facter.add(:hello) do
       setcode do
           "Hello, world"
       end
   end
   ```

2. 在客户端运行 Puppet。这会将 fact 同步到客户机：

```
# puppet agent --test
info: Retrieving plugin

notice: /File[/var/lib/puppet/lib/facter/hello.rb]/ensure: defi
content as '{md5}7314e71d35db83b563a253e741121b1d'

info: Loading downloaded plugin /var/lib/puppet/lib/facter/hell
info: Loading facts in hello
info: Loading facts in hello
info: Loading facts in hello
info: Loading facts in hello
info: Connecting to sqlite3 database: /var/lib/puppet/state/
clientconfigs.sqlite3

info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1297258039'

notice: Finished catalog run in 0.57 seconds
```

3. 通过直接运行 Facter 命令的方式检测 fact：

```
# facter hello
Hello, world
```

4. 现在你可以在一个 Puppet 的配置清单中应用这个自定义的 fact：

```
notify { $hello: }
```

5. 当你运行 Puppet，对自定义 fact 的引用将返回其对应的值：

```
notice: Hello, world
```

工作原理

Facter 内置的 facts 与我们刚刚创建的自定义 fact 相同的方式定义。 这种架构使添加和修改 facts 更为方便，并为你提供了一种在配置清单中读取主机信息的标准方法。

Facts 可以包含任何 Ruby 代码，语句块 setcode do ... end 中最后算出的值将作为 fact 的返回值。 例如，你可以做个更有用的 fact，下面的代码将返回当前登录的用户数：

```
Facter.add(:users) do
    setcode do
        %x{/usr/bin/who |wc -l}.chomp
    end
end
```

其输出是：

```
notice: 2 users logged in
```

更多用法

你可以扩展 facts 使用以创建一个完全“无节点定义（nodeless）”的 Puppet 配置： 换言之，Puppet 可以仅基于 facts 的结果决定将哪些资源应用到一台机器。Jordan Sissel 写了篇介绍这种方法的文章：http://www.semicomplete.com/blog/geekery/puppet-nodeless-configuration.html 。

在网络上有许多可用的自定义 facts 的例子，包括 Cosimo Streppone 撰写的关于“根据 IP 地址决定数据中心的位置”的文章，网址为：http://my.opera.com/cstrep/blog/puppet-custom-facts-and-master-less-puppet-deployment 。

在运行 **Puppet** 之前和之后执行命令

如果你希望在每次运行 Puppet 之前执行命令，可以在配置文件中使用 prerun_command 配置。 类似地，你也可以使用 postrun_command 配置运行 Puppet 之后需要执行的命令。 这种机制为 Puppet 与其他软件的集成提供了强大的钩子，甚至可以触发其他机器上的事件。

prerun 和 postrun 命令必须能成功运行（即其返回的状态码为 0），否则 Puppet 将报告一个错误。这可以让你通过 Puppet 的报告机制获得任何命令的错误报告。

操作步骤

在 puppet.conf 中设置 prerun_command 或 postrun_command 要执行的命令：

```
prerun_command  = /usr/local/bin/before-puppet-run.sh
postrun_command = /usr/local/bin/after-puppet-run.sh
```

更多用法

你可以使用 prerun 和 postrun 命令将 Puppet 与 Ubuntu 的 etckeeper 版本库整合起来。 **Etckeeper** 是一种用于跟踪 /etc 目录中文件变化的版本控制系统。 为了实现此功能，在 puppet.conf 配置文件中定义如下的执行命令：

```
prerun_command=/etc/puppet/etckeeper-commit-pre
postrun_command=/etc/puppet/etckeeper-commit-post
```

从 Shell 会话生成 Puppet 配置清单

> I object to being called a chess genius, because I consider myself to be an all around genius who just happens to play chess, which is rather different.
>
> — Bobby Fischer

并非所有人都是天才。如果你确切地知道安装一个应用程序或服务都需要做些什么的话， 你马上就可以创建 Puppet 的配置清单。尽管如此，你通常还是需要首先做些小小的试验， 比如找到你要安装的软件包、需要编辑哪些配置文件等等。 你可以使用 script 命令记录你的 Shell 会话，然后根据会话文件的记录内容开发 Puppet 的配置清单，这是个不错的方法。

但如果有一个工具能通过读取你的会话文件生成 Puppet 配置清单的话是不是更精彩呢？ 为了实现此功能，**cft** （读音为 'sift'）应运而生。 一旦你激活它，cft 监视你的 Shell 会话并记住你安装的任何软件包、任何服务的配置、 任何你创建或编辑的文件，等等。 当会话记录结束，cft 会生成一个重现你刚刚所做的所有改变的完整的 Puppet 配置清单。

准备工作

1. 当前完整的 cft 支持仅能用于 Red Hat/CentOS 发行版；针对 Debian/Ubuntu 发行版的完整的支持正在开发中，估计不久之后即可完成。 如果你正在使用 Red Hat/CentOS，只要安装 cft 即可：

   ```
   # yum install cft
   ```

2. 对于 Debian/Ubuntu 系统，请参考如下安装说明，网址为：http://fmtyewtk.blogspot.com/2011/01/porting-cft-to-debian.html 。

操作步骤

1. 在本例中我们将使用 cft 监视 NTP 安装的软件包并生成实现相同功能的配置清单。

   ```
   # cft begin ntp
   # apt-get install ntp
   Reading package lists... Done
   Building dependency tree
   Reading state information... Done
   Suggested packages:
     ntp-doc
   The following NEW packages will be installed:
     ntp
   0 upgraded, 1 newly installed, 0 to remove and 385 not upgradec
   ```

```
Need to get 517kB of archives.
After this operation, 1,323kB of additional disk space will be
Get:1 http://us.archive.ubuntu.com/ubuntu/ lucid/main ntp
1:4.2.4p8+dfsg-1ubuntu2 [517kB]
Fetched 517kB in 5s (101kB/s)
Selecting previously deselected package ntp.
(Reading database ... 135278 files and directories currently
installed.)
Unpacking ntp (from .../ntp_1%3a4.2.4p8+dfsg-1ubuntu2_i386.deb)
...
Processing triggers for man-db ...
Processing triggers for ureadahead ...
ureadahead will be reprofiled on next reboot
Setting up ntp (1:4.2.4p8+dfsg-1ubuntu2) ...
 * Starting NTP server ntpd

# vi /etc/ntp.conf
# service ntp restart
 * Stopping NTP server ntpd
[ OK ]
 * Starting NTP server ntpd

# cft finish ntp
# cft manifest ntp

class ntp {
    package { 'ntp':
        ensure =&gt; '1:4.2.4p8+dfsg-1ubuntu2'
    }

    service { 'ntp':
        enable =&gt; 'true',
        ensure =&gt; 'running'
    }

    file { '/etc/ntp.conf':
        group  =&gt; 'root',
        owner  =&gt; 'root',
        mode   =&gt; '0644',
        source =&gt; '/tmp/cft/ntp/after/etc/ntp.conf'
    }
}
```

工作原理

首先告诉 cft 开始记录系统的改变，并将其会话存储在 ntp 中?—?cft begin ntp 。

然后，当你安装 ntp 软件包时，cft 会记录这个事实。 软件包安装了服务的启动脚本，配置了在机器启动时启动服务，cft 同时也记录了这些。 最后，cft 注意到你编辑了 /etc/ntp.conf 文件，并保存了一份修改后的拷贝以备后用。

当你运行 cft finish ntp 命令，这会停止记录变化。 现在你可以使用 cft manifest ntp 命令生成与你的控制台会话等效的 Puppet 配置清单。

正如你看到的，生成的配置清单包括了 package 声明（由命令 apt-get install ntp 触发）：

```
package { 'ntp':
    ensure => '1:4.2.4p8+dfsg-1ubuntu2'
}
```

同时包括了再现包安装脚本作用的 service 声明，启动服务并设置开机启动：

```
service { 'ntp':
    enable => 'true',
    ensure => 'running'
}
```

这个声明是由于你手动配置了如下命令所生成的：

```
# service ntp start
# update-rc.d ntp defaults
```

配置清单的最后一部分封装了 ntp.conf 文件的改变。 cft 只知道你对这个文件做了改变，但不知道你具体做了哪些改变， 所以 cft 将修改后的整个文件做为一个拷贝，并使其可以通过 Puppet 分发这个文件：

```
file { '/etc/ntp.conf':
    group  => 'root',
    owner  => 'root',
    mode   => '0644',
    source => '/tmp/cft/ntp/after/etc/ntp.conf'
}
```

当你将此配置清单放入 Puppet，还需要从原始路径（/tmp/cft/ntp/after/etc/ntp.conf） 复制 ntp.conf 文件到你的模块树的适当位置，并根据这个位置修改 source 参数的值。

更多用法

cft 是快速生成 Puppet 配置清单原型的一个强大工具。 你可以找一台构建配置清单的主机，尽可能使用 cft 记录你的安装和配置过程， 并使用它对整个会话进行编码生成 Puppet 的配置清单。 虽然这还需要一些额外的编辑工作，但是比你从头开始编写配置清单要快得多。

从运行的系统上生成 Puppet 配置清单

除了使用 cft 从系统管理员的控制台会话生成 Puppet 配置清单以外， 还可以使用 **puppet resource** 从系统中已存在的配置生成 Puppet 配置清单。 例如，你可以使用 puppet resource 生成系统中所有用户的配置清单。 这对于生成工作系统的快照并将这些配置快速转换到 Puppet 是相当有用的。

操作步骤

1. 要生成指定用户的配置清单，请运行：

```
# puppet resource user john
user { 'john':
    password_min_age =&gt; '0',
    password_max_age =&gt; '99999',
    uid      =&gt; '1002',
    password =&gt; '!',
    gid      =&gt; '1001',
    groups   =&gt; ['git'],
    ensure   =&gt; 'present',
    comment  =&gt; 'John Arundel',
    home     =&gt; '/home/john',
    shell    =&gt; '/bin/bash'
}
```

2. 要生成指定服务的配置清单，请运行：

```
# puppet resource service ntp
service { 'ntp':
    ensure =&gt; 'running',
    enable =&gt; 'true'
}
```

3. 要生成指定软件包的配置清单，请运行：

```
# puppet resource package exim4
package { 'exim4':
    ensure =&gt; '4.71-3ubuntu1'
}
```

更多用法

你可以使用 puppet resource 检查 Puppet 每种可用的资源类型。 在上面的例子中，我们针对一个资源类型的具体实例生成了配置清单， 然而你也可以使用 puppet resource 导出一种资源类型的所有实例：

```
# puppet resource user
user { 'Debian-exim':
    ensure           => 'present',
    uid              => '117',
    gid              => '124',
    home             => '/var/spool/exim4',
    password         => '!',
    password_min_age => '0',
    password_max_age => '99999',
    shell            => '/bin/false'
}

user { 'avahi':
    ensure           => 'present',
    uid              => '104',
    gid              => '111',
    home             => '/var/run/avahi-daemon',
    password         => '*',
    password_min_age => '0',
    comment          => 'Avahi mDNS daemon,,,',
    password_max_age => '99999',
    shell            => '/bin/false'
}
...
```

这会产生许多输出！

使用 Puppet Dashboard

Puppet Dashboard 是一个管理 Puppet 安装的有用的工具，尤其对于大量的安装， 并且能够通过一个 Web 界面看到节点的信息和报告。 **Puppet Dashboard** 可以为你显示最近运行 Puppet 的节点，它们运行了多长时间， 是否有任何节点的错误报告，以及是否有一段时间内没有运行 Puppet 的节点等。

准备工作

1. 从 Puppet Labs 站点 http://www.puppetlabs.com/misc/download-options/ 下载 Puppet Dashboard 软件包并解压缩。 软件包中有一个安装说明文件 README.markdown，但你可能需要先安装以下依赖的包（部分或全部）：

```
# apt-get install -y build-essential irb libmysql-ruby
libmysqlclient-dev libopenssl-ruby libreadline-ruby mysql-serve
rake rdoc ri ruby ruby-dev
```

2. 为 Puppet Dashboard 应用程序创建一个 MySQL 数据库和用户（使用自己的口令）：

```
# mysql -uroot
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 39
Server version: 5.1.41-3ubuntu12.9 (Ubuntu)

Type 'help;' or '\h' for help. Type '\c' to clear the current
input statement.

mysql&gt; create database dashboard;
Query OK, 1 row affected (0.00 sec)

mysql&gt; grant all on dashboard.* to dashboard@localhost ident
by 'topsecret';
Query OK, 0 rows affected (0.01 sec)

mysql&gt; flush privileges;
Query OK, 0 rows affected (0.00 sec)
```

3. 复制 Puppet Dashboard 提供的样例文件 database.yml，并做适当的修改：

```
# cd puppetlabs-puppet-dashboard-071acf4
# cp config/database.yml.example config/database.yml
# vi config/database.yml
```

```
production:
  database: dashboard
  username: dashboard
  password: topsecret
  encoding: utf8
  adapter: mysql
```

4. 使用应用程序提供的 Rake 任务创建如下的初始化数据库：

```
# rake RAILS_ENV=production db:migrate
```

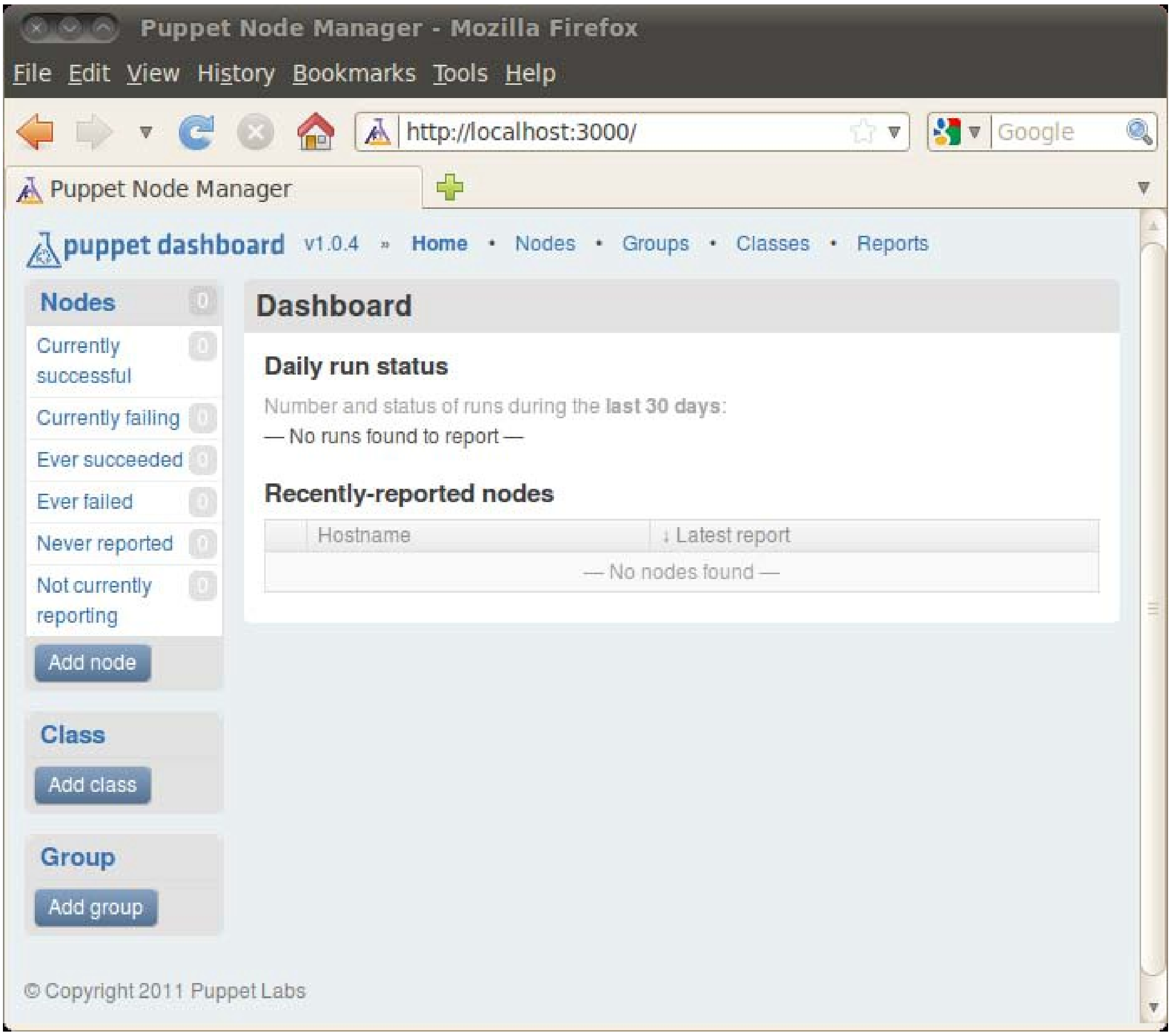
操作步骤

1. 启动内置的 Web 服务器：

```
# script/server -e production
=&gt; Booting WEBrick
=&gt; Rails 2.3.5 application starting on http://0.0.0.0:3000
=&gt; Call with -d to detach
=&gt; Ctrl-C to shutdown server
[2011-02-21 09:54:32] INFO WEBrick 1.3.1
[2011-02-21 09:54:32] INFO ruby 1.8.7 (2010-01-10) [i486-linux]
[2011-02-21 09:54:37] INFO WEBrick::HTTPServer#start: pid=1657
port=3000

Using a web browser, connect to localhost:3000
```

2. 如图所示，你应该看到 Puppet Dashboard 的界面：



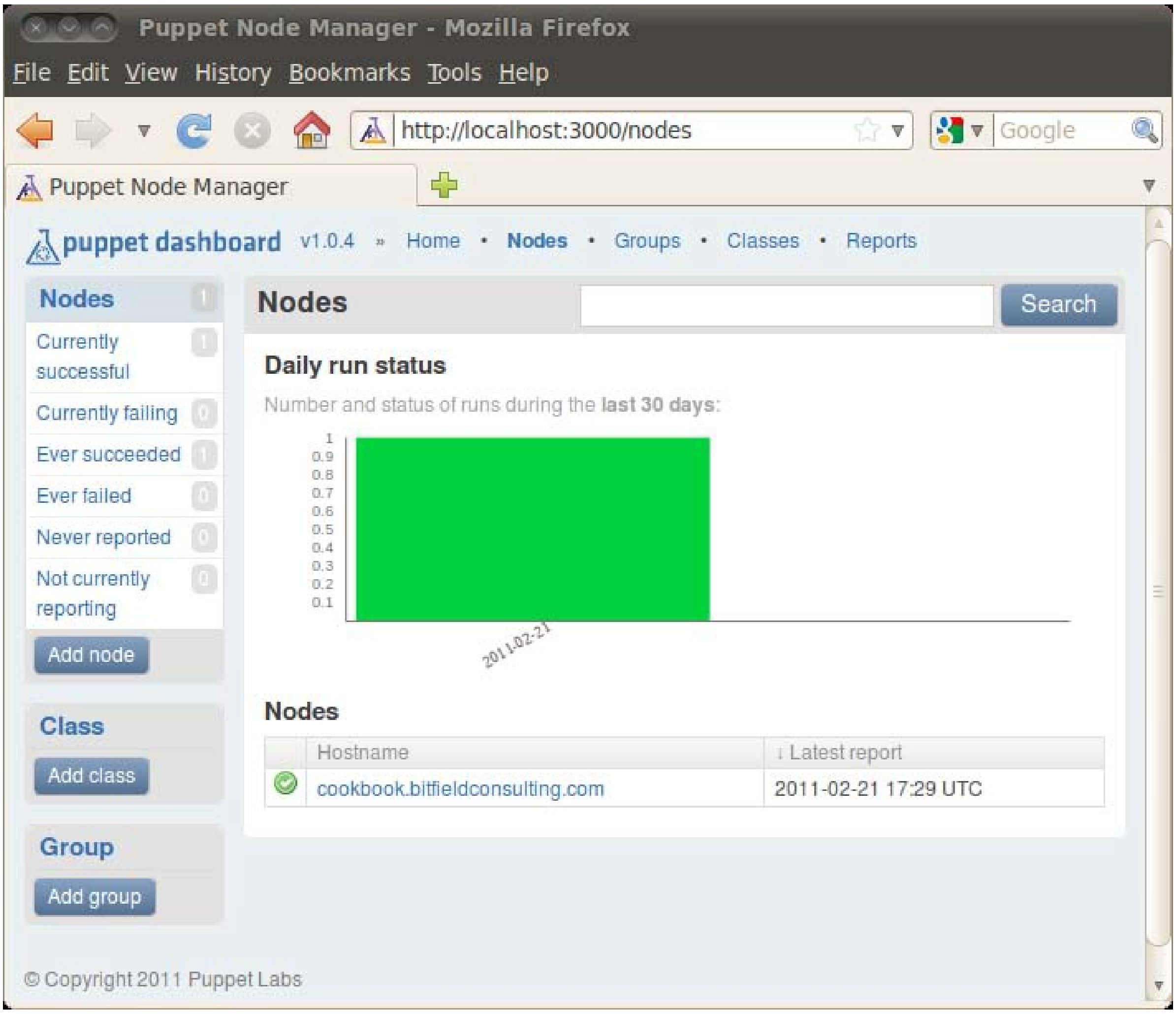
3. 现在你需要配置 Puppetmaster 向 Puppet Dashboard 发送报告。为了实现这个功能，你需要配置 puppet.conf 文件的 reports 参数，为其添加 http 报告：

```
reports = http,log
```

4. 重新启动 Puppetmaster 使新配置的报告生效。
5. 在节点上运行 Puppet：

```
# puppet agent --test
```

6. 在 Puppet Dashboard 界面中单击 **Nodes** 链接。如图所示，你应该看到表明 Puppet 运行成功的绿色区域：



工作原理

当在一个节点上运行了 Puppet，它会使用其报告装置向 Puppet Dashboard 发送报告。 Puppet Dashboard 会存储这些数据并利用这些数据显示所有节点上的 Puppet 活动的图表和摘要。

更多用法

你也可以使用 Puppet Dashboard 创建新的节点或类，并控制哪些节点需要包含哪些类。 实际上，它为你管理 Puppet 配置清单提供了一个 Web 接口， 所以你可以通过 Web 浏览器来编辑配置清单，而不是直接修改文本文件。 这是一个有吸引力的功能，尤其用于想让其他团队或部门的人能够管理他们自己的 Puppet 配置时。

为了实现 Puppet Dashboard 的这个功能，你需要配置 Puppet 使用 **external node classifier**； 这将在 使用外部节点分类器 一节中介绍。

参见本书

- 第 2 章的 生成报告 一节
- 第 2 章的 创建图形化报告 一节

- 本章的 使用外部节点分类器 一节

使用 Foreman

Foreman 是一个基于 Web 的类似于 Puppet Dashboard 的 Puppet 管理工具，但它更为雄心勃勃。 Foreman 不仅可以管理 Puppet 报告、节点和配置清单，而且可以为你供应新机器。 如果你需要自动创建大量的服务器或者你需要频繁地重建服务器，Foreman 将有助于你实现这一过程。

准备工作

1. 在你的系统中添加 Foreman 包的仓库，参考说明：http://theforeman.org/projects/foreman/wiki/Installation_instructions 。
2. 安装 Foreman 包：

```
# apt-get update
# apt-get install foreman
```

3. 系统将提示你选择一种数据库，请根据需要选择 mysql、pgsql 或 sqlite。
4. 根据选择的数据库，安装你的数据库所依赖的软件包：

```
# apt-get install foreman-mysql
# apt-get install foreman-pgsql
# apt-get install foreman-sqlite3
```

5. 复制文件 /etc/foreman/extras/puppet/foreman/files/foremanreport.rb 到你的 Puppet 自定义报告目录（通常是 /usr/lib/ruby/1.8/puppet/reports） 并将其重命名为 foreman.rb：

```
# cp /etc/foreman/extras/puppet/foreman/files/foreman-report.rb
    /usr/lib/ruby/1.8/puppet/reports/foreman.rb
```

6. 编辑 foreman.rb 文件设置你的 Foreman 服务器 URL：

```
# URL of your Foreman installation
$foreman_url="http://cookbook.bitfieldconsulting.com:3000"
```

7. 编辑你的 puppet.conf 添加 foreman 报告类型到已启用的报告类型列表：

```
[master]
reports = store,log,foreman
```

8. 重新启动 Puppetmaster 使新配置的报告生效。

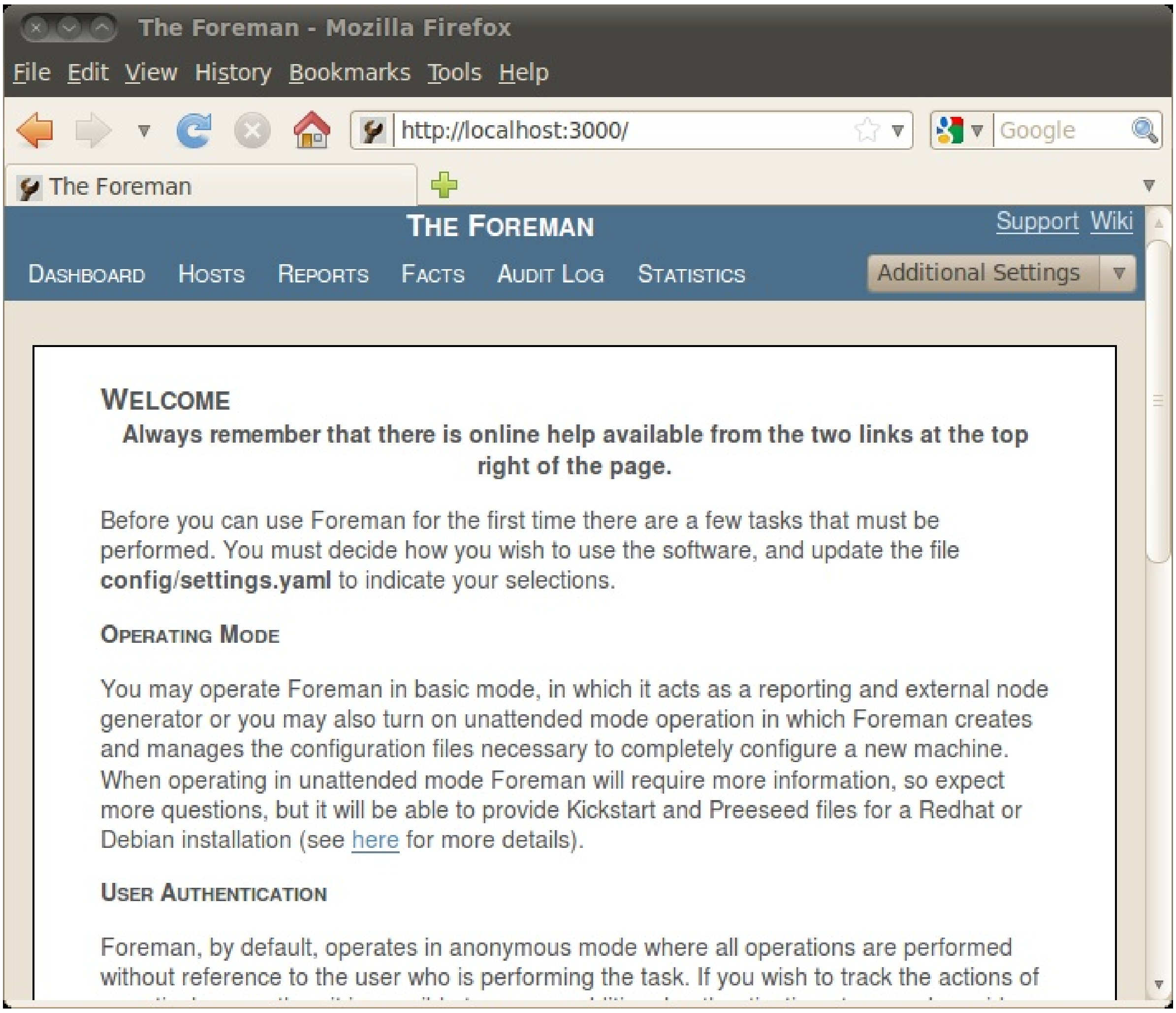
操作步骤

1. 启动 Foreman 服务器：

```
# /usr/share/foreman/script/server -e production
```

2. 使用你之前设置的 URL（http://cookbook.bitfieldconsulting.com:3000）浏览 Foreman 的 web 界面。

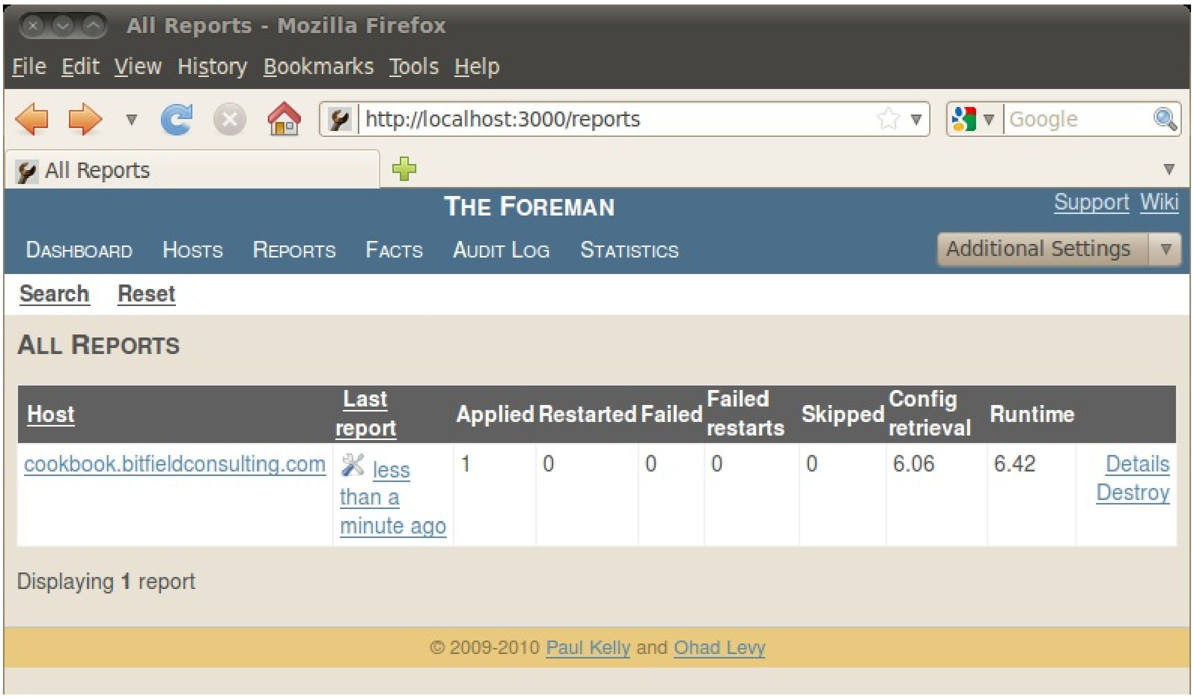
   你应该看到如下的 Foreman 初始欢迎界面：



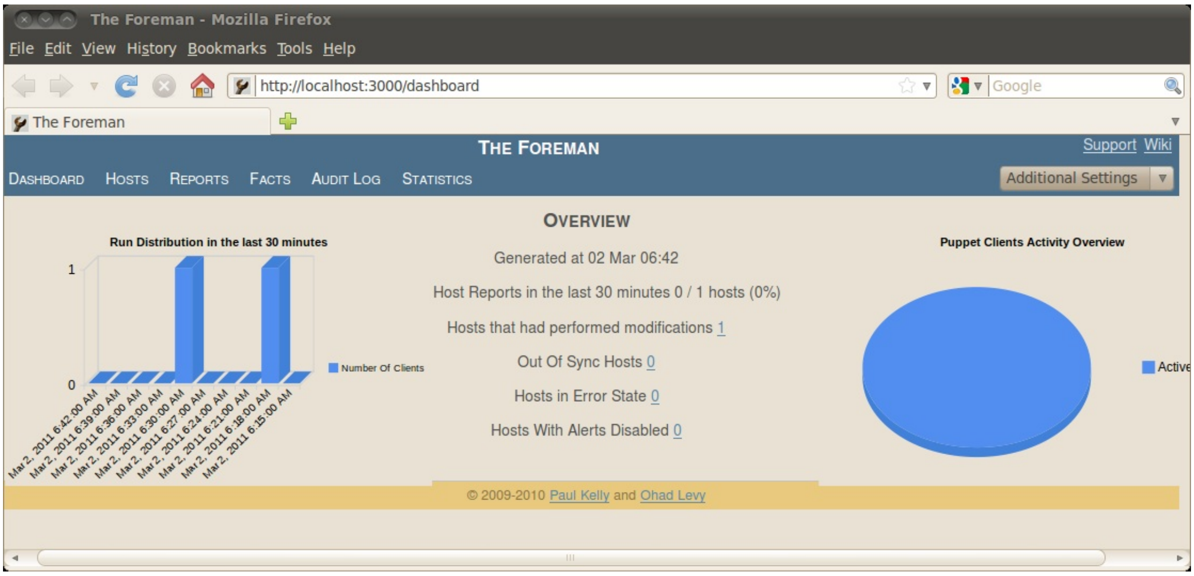
3. 现在，在一个客户端运行 Puppet：

```
# puppet agent --test
```

   在 Foreman 的 Web 界面中进入 **Reports** 标签页，如图所示，你会看到刚刚运行的客户报告：



4. 进入 Dashboard 标签页你会看到一个针对所有客户的 **OVERVIEW** （本例中只有一个客户）。



更多用法

此处我们仅仅接触到了 Foreman 的一些基本功能。既然已经运行了一个配置好的 Foreman， 你就可以尝试不同的报告、图表、以及 Foreman 提供的其他信息。 当你有很多主机需要管理，并希望看到 Puppet 如何跨越整个网络的运行统计时， 这些信息会变得更有价值。

Foreman 的另一个主要特性是供应：它可以使用 PXEboot 和 Kickstart 从头创建物理服务器或虚拟服务器，对 Puppet 证书进行自动签名，并运行 Puppet 将机器带入生产状态。 欲了解更多有关如何实现的详细信息，请查阅 Foreman 文档：http://theforeman.org/projects/foreman/wiki/Unattended_installations 。

> 如果你决定将 Foreman 用于生产环境，就值得为其设置 Apache 虚拟主机，而不是使用 Webrick 带动 Foreman。 Webrick 对测试环境很有用，但它不是真正的生产级别的 Web 服务器。

使用 MCollective

Marionette Collective（简称为 **MCollective**）是一个系统管理工具。MCollective 可以在大量服务器上并行运行命令，它采用广播架构，所以，你可以使用它管理一个大型网络而不需要一个中央主服务器或资产数据库。

每台服务器都运行一个 MCollective 守护进程监听请求，并在本地执行命令或返回有关服务器的信息。这也可以用来过滤目标服务器列表。例如，你可以使用 MCollective 在符合特定条件的所有服务器上执行给定的命令。

你可以考虑将 MCollective 作为 Puppet 的一个补充（尽管它也可以与 Chef 或其他配置管理系统协同工作）。例如，你配置一个新节点的过程可能需要改变其他机器上的防火墙配置，对数据库服务器授予适当的权限，等等。这仅使用 Puppet 是不太容易做到的。虽然你可以使用 Shell 脚本和 SSH 自动化执行特定的工作，但是 MCollective 提供了解决这个普遍问题的强大而灵活的方式。

准备工作

1. MCollective 使用 ActiveMQ 消息代理框架（实际上，可以使用任何 STOMPcompliant 中间件，但 ActiveMQ 是一个受欢迎的选择），ActiveMQ 需要 Java 运行环境，如果你的系统还没有安装 Java，先安装它：

```
# apt-get install gcj-4.4-jre-headless
```

2. 到 ActiveMQ 下载页面 http://activemq.apache.org/download.html 下载 “Unix 发布” 最近的稳定版 tar 包。

3. 安装 stomp gem ：

```
# gem install stomp
```

4. 到 http://www.puppetlabs.com/misc/download-options/ 下载 MCollective 最近的稳定版 .deb 包。

5. 安装已下载的 .deb 包：

```
# dpkg -i mcollective_1.0.1-1_all.deb mcollective-client_1.0.1-
all.deb mcollective-common_1.0.1-1_all.deb
```

6. 从 MCollective 的下载页面下载与 .deb 版本相同的 tar 包 （因为其中包含了 ActiveMQ 配置文件样例）。

7. 编辑 MCollective 的 server.cfg 文件：

```
# vi /etc/mcollective/server.cfg
```

8. 将参数 plugin.stomp.host 设置为你的服务器名（即你运行 ActiveMQ 的服务器）：

```
plugin.stomp.host = cookbook.bitfieldconsulting.com
```

9. 对 MCollective 的 client.cfg 文件做同样的设置：

```
# vi /etc/mcollective/client.cfg
```

10. 解压缩 MCollective 的 tar 包并复制 ActiveMQ 的配置文件样例到 /etc/mcollective：

```
# tar xvzf mcollective-1.0.1.tgz
# cp mcollective-1.0.1/ext/activemq/examples/single-broker/acti
   /etc/mcollective
```

11. 编辑这个配置文件设置 mcollective 用户的口令与 server.cfg 中的相同：

```
# vi /etc/mcollective/activemq.xml
```

12. 解压缩 ActiveMQ 的 tar 包，用指定的配置文件启动 ActiveMQ 的服务：

```
# tar xvzf apache-activemq-5.4.2-bin.tar.gz
# apache-activemq-5.4.2/bin/activemq start xbean:/etc/mcollecti
activemq.xml

INFO: Using default configuration
(you can configure options in one of these file: /etc/default/
activemq /root/.activemqrc)

INFO: Invoke the following command to create a configuration fi
bin/activemq setup [ /etc/default/activemq | /root/.activemqrc

INFO: Using java '/usr/bin/java'

INFO: Starting - inspect logfiles specified in logging.properti
and log4j.properties to get details

INFO: pidfile created : '/root/apache-activemq-5.4.2/data/
activemq.pid' (pid '3322')
```

13. 启动 MCollective 的服务：

```
# service mcollective start
Starting mcollective: *
```

操作步骤

1. 使用如下命令检查 MCollective 和 ActiveMQ 是否启动且正常运行：

```
# mc-ping
cookbook                                    time=68.82 ms

---- ping statistics ----
1 replies max: 68.82 min: 68.82 avg: 68.82
```

2. 如果未看到任何结果输出，检查 mcollectived 守护进程是否已运行，并检查用于 ActiveMQ 的 Java 进程是否已运行。

3. 针对你的机器运行 mc-inventory 查看 MCollective 知道的关于 cookbook 机器的信息：

```
# mc-inventory cookbook
Inventory for cookbook:

    Server Statistics:
                          Version: 1.0.1
                       Start Time: Mon Mar 07 11:44:53 -0700 2011
                      Config File: /etc/mcollective/server.cfg
                       Process ID: 4220
                   Total Messages: 14
          Messages Passed Filters: 6
                Messages Filtered: 5
                     Replies Sent: 5
             Total Processor Time: 0.8 seconds
                      System Time: 0.47 seconds

    Agents:
             discovery rpcutil
    Configuration Management Classes:

    Facts:
          mcollective =&gt; 1
```

4. 通过在 /etc/mcollective/facts.yaml 中添加如下的代码片段为服务器创建一个新的自定义 fact：

```
purpose: webserver
```

5. 现在使用 MCollective 查找所有匹配这个 fact 的机器：

```
# mc-find-hosts --with-fact purpose=webserver
cookbook
```

工作原理

MCollective 是一个广播框架；当你发出一个像 mc-find-hosts 这样的请求时，MCollective 就为所有请求的客户端发送消息：“有没有客户户匹配这个过滤器呀？” 所有匹配过滤器的客户端都会发送一个回复，MCollective 会将这些回复收集到一起并为你输出这些信息。

你可以为特定的任务安装大量的插件和代理（例如，运行 Puppet）。 这些插件和代理都是安装在客户端上的，MCollective 处理发送命令到所有匹配机器所涉及的通信， 并整理勘校其任何结果。

更多用法

尽管我们只涉及了使用 MCollective 的一些基本步骤，但是它显然是一个功能强大的工具， 它既可以收集有关服务器的信息，也可以通过选择 fact 针对一个服务器列表执行命令。 例如，你可以获得一个过去 24 小时没有运行 Puppet 的机器列表。 又如，你可以对所有 Web 服务器或所有 x86_64 架构的机器执行一系列动作。

MCollective 本身只为这类应用提供了一个框架。对于不同的应用有各种各样的插件可用， 而且编写自己的插件也是很容易的。在下面的例子中，我们将使用 package 插件，此插件允许你查询和操作包。

安装 **MCollective** 插件

1. 从 GitHub 克隆 MCollective 插件仓库：

```
# git clone https://github.com/puppetlabs/mcollective-plugins.g
```

2. 复制插件文件到适当的目录：

```
# cd mcollective-plugins
# cp agent/package/mc-package /usr/bin
# cp agent/package/puppet-package.rb \
     /usr/share/mcollective/plugins/mcollective/agent/package.r
# cp agent/package/package.ddl \
     /usr/share/mcollective/plugins/mcollective/agent
```

3. 重新启动 MCollective：

```
# service mcollective restart
```

4. 运行 mc-inventory 检查该插件是否出现在 Agents 列表中：

```
# mc-inventory cookbook

Inventory for cookbook:
    Server Statistics:
                       Version: 1.0.1
                    Start Time: Tue Mar 08 08:28:29 -0700 2011
                   Config File: /etc/mcollective/server.cfg
                    Process ID: 6047
               Total Messages: 1
      Messages Passed Filters: 1
            Messages Filtered: 0
                 Replies Sent: 0
         Total Processor Time: 0.04 seconds
                  System Time: 0.02 seconds

    Agents:
        discovery        package        rpcutil

    Configuration Management Classes:

    Facts:
        mcollective =&gt; 1
        purpose =&gt; webserver
```

5. 试着执行 mc-package 命令检验如下命令是否正常工作：

```
# mc-package status apache2
Do you really want to operate on packages unfiltered? (y/n): y
 * [ =========================================================&

cookbook                         version = apache2-2.2.14-5u

---- package agent summary ----
          Nodes: 1 / 1
       Versions: 1 * 2.2.14-5ubuntu8.4
   Elapsed Time: 0.58 s
```

package 代理提供了一个强大的方法用于在你的整个网络（或特定的机器）上检查包版本、安装或更新所需的软件包。有关此插件和其他 MCollective 插件的更多细节，请查看 wiki 页面：http://projects.puppetlabs.com/projects/mcollective-plugins/wiki 。

有关 MCollective 的更多信息，请访问其主页：http://docs.puppetlabs.com/mcollective/ 。

使用公共模块

> Plagiarize, plagiarize, plagiarize / Only be sure always to call it, please *research*.
>
> *Lobachevsky* — Tom Lehrer

如果你对自己编制的代码有疑问，使用他人的吧。在许多情况下， 当你要写一个 Puppet 模块来管理一些软件或服务时，不必从头开始编写。 对许多流行的应用程序，在 **Puppet Forge** 站点上都提供了社区贡献的模块。 有时，一个社区模块正是你所需要的，你可以下载并马上开始使用它。 若社区提供的模块不能满足你的需求，你可以对其进行一些修改，以适应你的特定需求和环境。

如果你是个 Puppet 的新手，能从一些现有的代码开始对你将是一个很大的帮助。但另一方面， 社区模块往往编写得尽可能通用和便携，为了实现此目的所需的额外代码可能使其更难理解。

一般地，在你未阅读和理解 Puppet Forge 提供的模块代码之前， 我建议你不要将其作为‘活动的（drop-in）’模块源来部署。 Puppet Forge 提供的模块为你的 Puppet 基础设施引入了一个外部的依赖， 但它并不能帮你提升对 Puppet 的理解也不利于提升你使用 Puppet 的经验。 相反，我会用它作为一个获取灵感的源泉，从这方面讲它还是很有帮助的。 从 Puppet Forge 获取的模块应该是一个起点，你可以用它作为基础，制定和完善自己的模块。

要知道，一个给定的模块可能无法在你的 Linux 发行上工作。请检查模块自带的 README 文件，查看这个模块是否支持你的操作系统。

准备工作

1. 使用 Puppet Forge 模块的最简单方法是安装 puppet-module 工具：

```
# gem install puppet-module
Fetching: puppet-module-0.3.2.gem (100%)
****************************************************************

Thank you for installing puppet-module from Puppet Labs!
* Usage instructions: read "README.markdown" or run `puppetmodu
  usage`
* Changelog: read "CHANGES.markdown" or run `puppet-module
  changelog`
* Puppet Forge: visit http://forge.puppetlabs.com/

****************************************************************

Successfully installed puppet-module-0.3.2
1 gem installed
Installing ri documentation for puppet-module-0.3.2...
Installing RDoc documentation for puppet-module-0.3.2...
```

2. 运行 puppet-module 查看其可用的任务：

```
# puppet-module
Tasks:
  puppet-module build [PATH_TO_MODULE]
    # Build amodule for release
  puppet-module changelog
    # Display the changelog for this tool
  puppet-module changes [PATH_TO_MODULE]
    # Show modified files in an installed m...
  puppet-module clean
    # Clears module cache for all repositories
  puppet-module generate USERNAME-MODNAME
    # Generate boilerplate for a new module
  puppet-module help [TASK]
    # Describe available tasks or one speci...
  puppet-module install MODULE_NAME_OR_FILE [OPTIONS]
    # Install a module (eg, 'user-modname')...
  puppet-module repository
    # Show currently configured repository
  puppet-module search TERM
    # Search the module repository for a mo...
  puppet-module usage
    # Display detailed usage documentation ...
  puppet-module version
    # Show the version information for this...

Options:
  -c, [--config=CONFIG] # Configuration file
                        # Default: /etc/puppet/puppet.conf
```

操作步骤

在本例中，我们将使用 puppet-module 查找并安装一个管理 Tomcat 应用程序服务器的模块。

1. 查找合适的模块：

```
# puppet-module search tomcat
=====================================
Searching http://forge.puppetlabs.com
-------------------------------------
2 found.
--------
camptocamp/tomcat (0.0.1)
jeffmccune/tomcat (1.0.1)
```

2. 在本例中，我们将安装 Jeff McCune 版本的 tomcat 模块：

```
# cd /etc/puppet/modules
# puppet-module install jeffmccune/tomcat
Installed "jeffmccune-tomcat-1.0.1" into directory:
jeffmccune-tomcat
```

3. 现在就可以在你的配置清单中使用这个模块了：查看其源代码将向你展示这是如何实现的。

工作原理

puppet-module 工具使用简单地自动化过程从 Puppet Forge 站点搜索和下载模块。 你也可以浏览该网站查询可用的模块： http://forge.puppetlabs.com/ 。

更多用法

并非所有公开可用的模块都保存在 Puppet Forge 上。其他的存放位置是在 GitHub 上：

- https://github.com/camptocamp
- https://github.com/example42

Dean Wilson 在他的 Puppet Cookbook 站点： http://puppetcookbook.com/ 维护了一个优秀的 Puppet 的仓库，还包括一些技巧和处方。

使用外部节点分类器

当 Puppet 运行在一个节点上，它需要知道这个节点应该应用了哪些类。 例如，如果这是一个 web 服务器节点，它可能需要包含一个 apache 类。 将类映射到节点的一种简单方法是在配置清单里声明，例如下面是一个 nodes.pp 文件的例子：

```
node web1 {
    include apache
}
```

另外，你可以使用 外部节点分类器（**external node classifier**，**ENC**）来实现这个工作。 一个外部节点分类器是任何可执行程序，它可以接受一个节点的名称，并返回该节点的类列表。 例如，这可能是一个简单的 shell 脚本， 或者是对一个可以决定如何映射类到节点的更复杂的程序或 API 的封装。

准备工作

在你的 puppet.conf 配置文件中设置如下的参数：

```
[master]
external_nodes = /usr/local/bin/puppet_node_classifier
node_terminus  = exec
```

操作步骤

1. 创建如下的简单脚本 /usr/local/bin/puppet_node_classifier：

```
#!/bin/bash
if [ "$1" == "cookbook.bitfieldconsulting.com" ]; then
  cat &lt;&lt;"END"
---
classes:
  - admin::sudoers
  - admin::exim
  - puppet
  - nagios::target

environment: production
parameters:
  location: Bitfield HQ
END

else
  exit 1
fi
```

2. 为该脚本添加可执行权限：

```
# chmod 755 /usr/local/bin/puppet_node_classifier
```

3. 运行 Puppet：

```
# puppet agent --test
info: Retrieving plugin
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1299677816'
notice: Finished catalog run in 1.19 seconds
```

工作原理

Puppet 调用你在 puppet.conf 中由参数 external_nodes 指定的脚本， 并传递节点的名称作为命令行参数。在本例的脚本中，我们检查这个参数， 如果它与 cookbook.bitfieldconsulting.com 相同，将为其输出一个 Puppet 所需的 YAML 格式的类列表。 否则，此脚本将返回退出状态码 1（向 Puppet 指出该节点没有被发现）。

该脚本还设置了 environment 的值（参见 使用不同的环境 一节中关于这个参数的解释）。 同时将变量 location 设置成了 Bitfield HQ?—?此变量是个自定义变量，对于 Puppet 并无特殊含义， 但是由于在 ENC 中定义的变量是顶级范围的变量，可以在你的配置清单里引用此处定义的变量， 所以，例如你可以使用它决定 DNS 解析器的设置。你可以在此设置任何你需要的变量。

显然，这个脚本并不十分有用，因为它只是输出了一个预先确定的类列表。 一个更复杂的脚本可能会检查数据库，从中查找节点所需的类列表， 或从一个哈希或外部文本文件中查找节点所需的类列表。 我希望，这个例子足以让你开始写出自己的更复杂的外部节点分类器。

有关外部节点分类器的详细信息，请参考：
http://docs.puppetlabs.com/guides/external_nodes.html 。

更多用法

使用外部分类器的主要用途是使 Puppet 可以连接 LDAP 目录服务。 许多大型组织都有个 LDAP 基础设施，你可以设置 Puppet，使它可以从 LDAP 目录服务获取信息， 并且其他 LDAP 客户也可以通过 Puppet 获得由其管理的节点信息。

欲了解更多有关如何做到这一点的详细信息，请访问 “Puppet 和 LDAP” 页面：
http://projects.puppetlabs.com/projects/puppet/wiki/LDAP_Nodes 。

此功能也可用于 Puppet Dashboard 和 Foreman 通过 Web 界面管理节点和类之间的关系?—?他们将以外部节点分类器来处理。

创建自定义的资源类型

该到你发挥创意的时间了。你已经知道了各种不同的 Puppet 资源类型： 包（package）, 文件（file）、用户（user），等等。 通常情况下，你既可以组合使用这些内置资源类型做你需要做的一切， 又可以通过一个自定义 define 作为一种资源（以内置资源同样的方式）来使用 （参见第 4 章 书写更优质的配置清单 中有关 define 的内容）。

但是，如果你需要创建自己的资源类型，Puppet 也可以很容易地实现。 原生的资源类型都是使用 Ruby 书写的，为了创建自己的资源类型，你需要对 Ruby 有一个基本的了解。

让我们重新回顾一下 资源类型（**type**）和 提供者（**provider**）之间的区别。 资源类型描述了一个资源和它可拥有的参数（例如，package 类型）。 提供者则告诉 Puppet 如何针对特定的平台或情况去实现一个资源 （例如，apt/dpkg 提供者为 Debian/Ubuntu 系统实现 package 资源）。

一种类型（如：package）可以有多个提供者（如：apt、yum、fink 等等）。 如果你声明一个资源时没有指定提供者，Puppet 会根据环境选择一个最合适的提供者。

在本节中，我们将看到如何创建一个管理 Git 仓库的自定义资源类型； 在下一节，我们将编写一个实现这种资源类型的提供者。

准备工作

1. 在你的 puppet.conf 文件中启用 pluginsync（若还未启用）：

```
[main]
pluginsync = true
```

2. 在你的 Puppet 仓库中，为你的插件和类型创建一个自定义模块（若还不存在）：

```
# cd /etc/puppet/modules
# mkdir custom
```

3. 在这个模块中，创建 lib/puppet/type 目录：

```
# cd custom
# mkdir -p lib/puppet/type
```

操作步骤

在 type 目录中创建一个名为 gitrepo.rb 的文件，其内容如下：

```
Puppet::Type.newtype(:gitrepo) do
    ensurable

    newparam(:source) do
        isnamevar
    end

    newparam(:path)
end
```

工作原理

第一行注册一个名为 gitrepo 的新类型：

```
Puppet::Type.newtype(:gitrepo) do
```

ensurable 行确保自动给出该类型的属性（与 Puppet 内置的资源类似）：

```
    ensurable
```

现在，我们将给出此类型的一些参数。就目前而言，我们所需要的参数分别是：source 参数用于指定 Git 仓库源的 URL；path 参数用于告诉 Puppet 要在文件系统中的什么位置创建仓库。

```
    newparam(:source) do
        isnamevar
    end
```

isnamevar 声明告诉 Puppet 参数 source 是此类型的 namevar。 因此当你声明这个资源的实例时，你给出的任何名字将被视为 source 的值。例如：

```
gitrepo { "git://github.com/puppetlabs/puppet.git":
    path => "/home/john/work/puppet",
}
```

最后，我们添加 path 参数：

```
    newparam(:path)
```

更多用法

一旦你熟悉了创建自己的资源类型的方法，你就可以使用自定义的资源类型替换复杂的 exec 资源， 这会使你的配置清单更具可读性。 然而，通过对自定义资源类型的代码添加一些文档和参数校验使其更强壮更具可重用性是一个好主意。

文档

我在上面故意举了一个简单的例子，但是当你要为生产环境开发真正的自定义类型时， 你应该加入文档字符串描述类型及其参数的用途。例如：

```
Puppet::Type.newtype(:gitrepo) do
    @doc = "Manages Git repos"

    ensurable

    newparam(:source) do
        desc "Git source URL for the repo"
        isnamevar
    end

    newparam(:path) do
        desc "Path where the repo should be created"
    end
end
```

校验

当某人试图向资源传递错误的值时，你可以使用参数校验（validate）生成有用的错误信息。 例如，你可以校验要创建仓库的目录是否已真实存在：

```
  newparam(:path) do
    validate do |value|
      basepath = File.dirname(value)
      unless File.directory?(basepath)
        raise ArgumentError , "The path %s doesn't exist" %basepatl
      end
    end
  end
```

你也可以为参数指定一个允许的取值列表，例如：

```
newparam(:breakfast) do
    newvalues(:bacon, :eggs, :sausages)
end
```

创建自定义的提供者

在上一节，我们创建了一个新的名为 gitrepo 的自定义资源类型并告诉 Puppet 此类型需要携带两个参数，分别为 source 和 path。 然而到目前为止，我们还没有告诉 Puppet 如何检出仓库，即如何创建这种类型的具体实例。 这正是提供者（provider）的用武之地。

正如我们之前看到的，一个类型经常会有几种可能的提供者。在本例中， 对一个 Git 仓库进行实例化仅有一种明智的方法，所以我们只需一个提供者：git。 如果你想要扩展这个自定义类型（将其称之为 repo 而非 gitrepo）， 不难想象只要针对不同类型的仓库创建若干不同的提供者即可，例如：git、svn、cvs 等等。

准备工作

1. 在你的 custom 模块的 lib/puppet 目录中, 创建一个名为 provider/gitrepo 的子目录：

```
# mkdir -p lib/puppet/provider/gitrepo
```

2. 在 gitrepo 目录中，使用如下内容创建一个名为 git.rb 的文件：

```
require 'fileutils'

Puppet::Type.type(:gitrepo).provide(:git) do
    commands :git =&gt; "git"

    def create
        git "clone", resource[:source], resource[:path]
    end

    def exists?
        File.directory? resource[:path]
    end
end
```

操作步骤

1. 在 Puppet 配置清单中添加如下代码为新的资源类型 gitrepo 创建一个实例：

```
gitrepo { "https://github.com/puppetlabs/puppet.git":
    path   =&gt; "/tmp/puppet",
    ensure =&gt; present,
}
```

2. 现在运行 Puppet，你的新类型将被加载并进行了实例化：

```
# puppet agent --test
info: Retrieving plugin
notice: /File[/var/lib/puppet/lib/puppet]/ensure: created
notice: /File[/var/lib/puppet/lib/puppet/provider]/ensure: crea

notice: /File[/var/lib/puppet/lib/puppet/provider/gitrepo]/ensu
created

notice: /File[/var/lib/puppet/lib/puppet/provider/gitrepo/git.r
ensure: defined content as '{md5}a12870d89a4b517e48fe417ce2e12a

notice: /File[/var/lib/puppet/lib/puppet/type]/ensure: created

notice: /File[/var/lib/puppet/lib/puppet/type/gitrepo.rb]/ensur
defined content as '{md5}90d5809e1d01dc9953464e8d431c9639'

info: Loading downloaded plugin /var/lib/puppet/lib/puppet/
provider/gitrepo/git.rb

info: Loading downloaded plugin /var/lib/puppet/lib/puppet/type
gitrepo.rb

info: Redefining gitrepo in Puppet::Type
info: Caching catalog for cookbook.bitfieldconsulting.com
info: Applying configuration version '1299850325'

notice: /Stage[main]//Node[cookbook]/Gitrepo[https://github.com
puppetlabs/puppet.git]/ensure: created

notice: Finished catalog run in 74.43 seconds
```

注意：由于 Puppet 的一个错误，当你首次创建新类型的实例时，可能需要两次运行 puppet agent：第一次加载类型的定义，第二次才真正创建实例。 如果你看到如下的信息：

```
err: /Stage[main]//Node[cookbook]/Gitrepo[https://
github.com/puppetlabs/puppet.git]: Could not
evaluate: No ability to determine if gitrepo exists
```

就意味着你正遭遇此错误带来的困扰?—?别急，再次运行 Puppet 即可正常工作。 当你读到本书的出版物时，这个错误很可能已经被修复。

工作原理

首先我们为 gitrepo 类型注册一个资源类型的提供者：

```
Puppet::Type.type(:gitrepo).provide(:git) do
```

当你在配置清单中声明此类型的一个实例时，Puppet 会先检查是否有已经存在的实例：

```
def exists?
    File.directory? resource[:path]
end
```

Puppet 会调用我们实现的 exists? 方法来做这种检查。 如果已有一个匹配实例 path 参数的目录存在，它返回 true。

如果 exists? 返回 true，那么 Puppet 将不会采取进一步的行动， 否则 Puppet 将通过调用 create 方法试图创建这个资源：

```
def create
    git "clone", resource[:source], resource[:path]
end
```

在这种情况下，create 方法会执行 git clone ，这会将原始仓库 （由 source 参数指定）克隆到由 path 参数指定的目录。

更多用法

你已经看到 Puppet 的自定义类型和提供者的强大之处。 实际上，他们可以做任何事情?—?至少是 Ruby 可以做的任何事情。 如果在你管理的某一部分基础设施中，使用了复杂的 define 和 exec 资源， 你就应该考虑将它们替换为自定义资源类型。 实际上，在你创建自定义类型之前可以先环顾一下周围是否已经有人实现了你需要的自定义资源类型。

此处我举的例子比较简单，你还有更多有关书写自定义类型的内容需要学习。 如果你要分发代码以供他人使用（或者，即使你不分发代码），在代码中包含必要的测试是一个好主意。

Puppet Labs 有一些有关开发自定义类型的有用页面：
http://docs.puppetlabs.com/guides/custom_types.html 和
http://projects.puppetlabs.com/projects/1/wiki/Development_Practical_Types 。 有关如何编写符合 Puppet Labs 标准的测试信息，请参考
http://projects.puppetlabs.com/projects/1/wiki/Development_Writing_Tests 。

James Turnbull 为自定义类型的开发撰写了一篇相当不错的易于遵循的介绍文章 “Creating Puppet types and providers is easy…”，其地址为：
http://www.kartar.net/2010/02/puppet-types-and-providers-are-easy/ 。

Dean Wilson 也提供了一个非常有启发性的例子，用于管理 APT 资源：
https://github.com/deanwilson/puppet-aptsourced 。